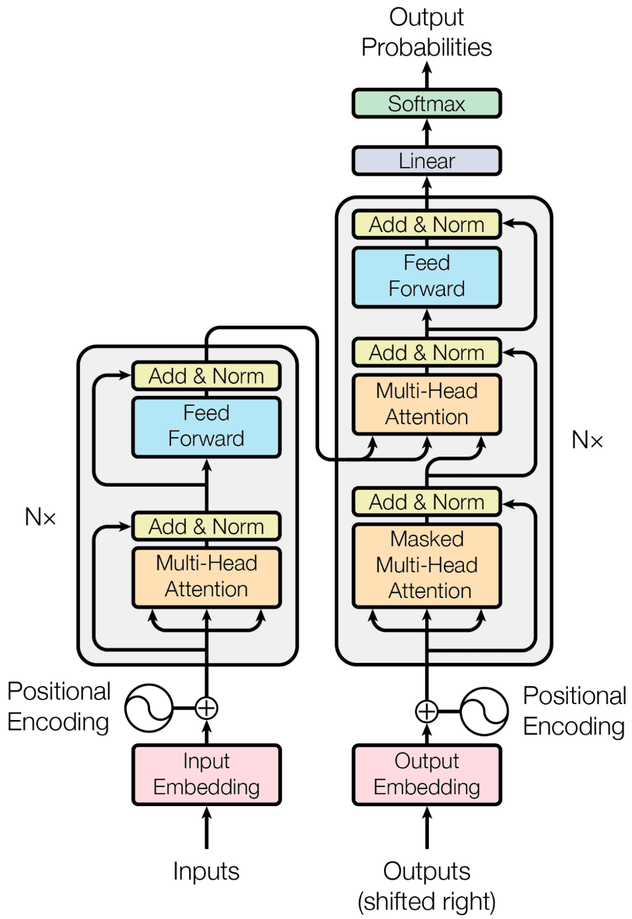
Efficient Large Language Model Deployment with Quantization

Guangxuan Xiao

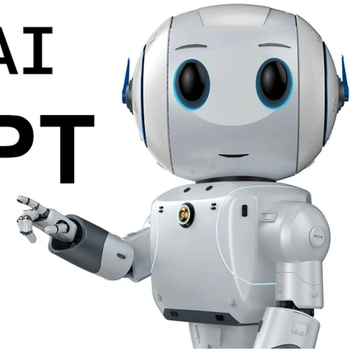
MIT

Large Language Models (LLMs) are Powerful

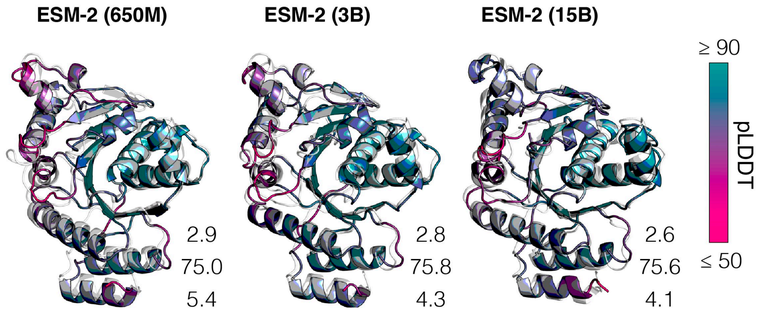


Transformer-based

OpenAI ChatGPT



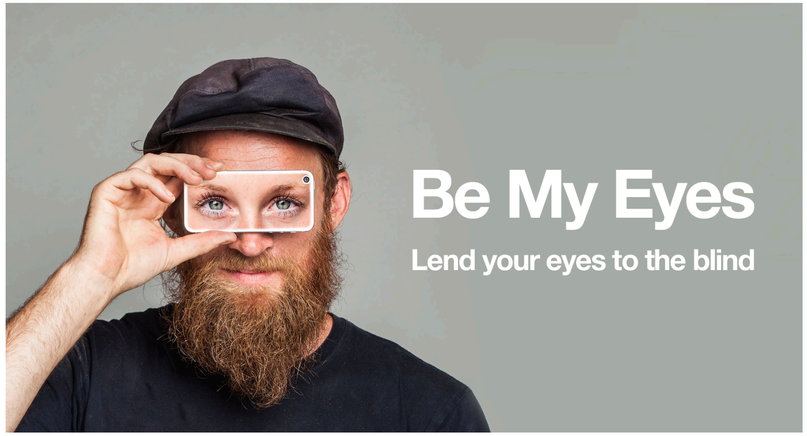
ChatBots



Scientific Discovery

GitHub Copilot

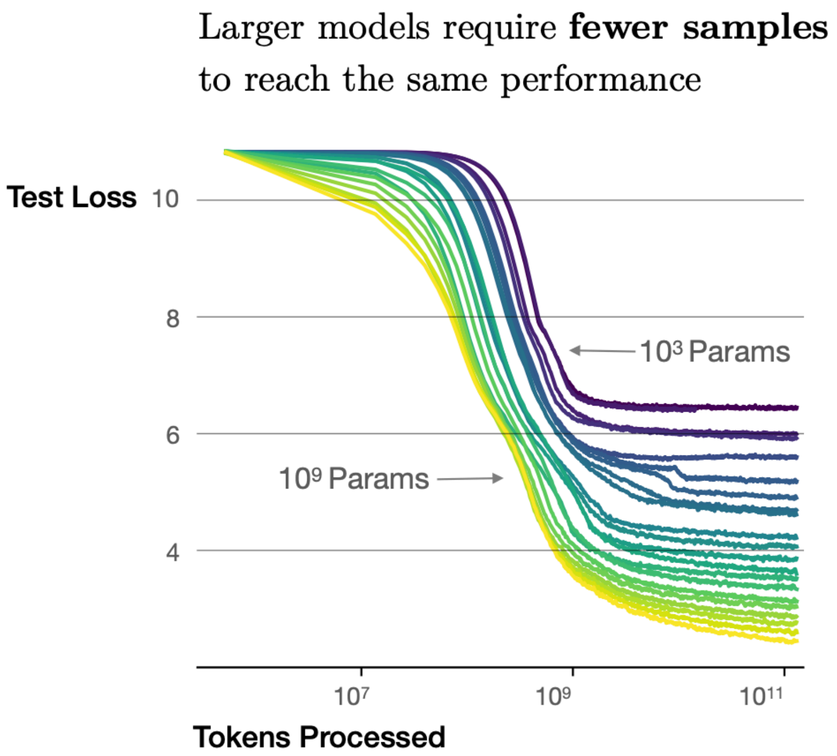
Software Development

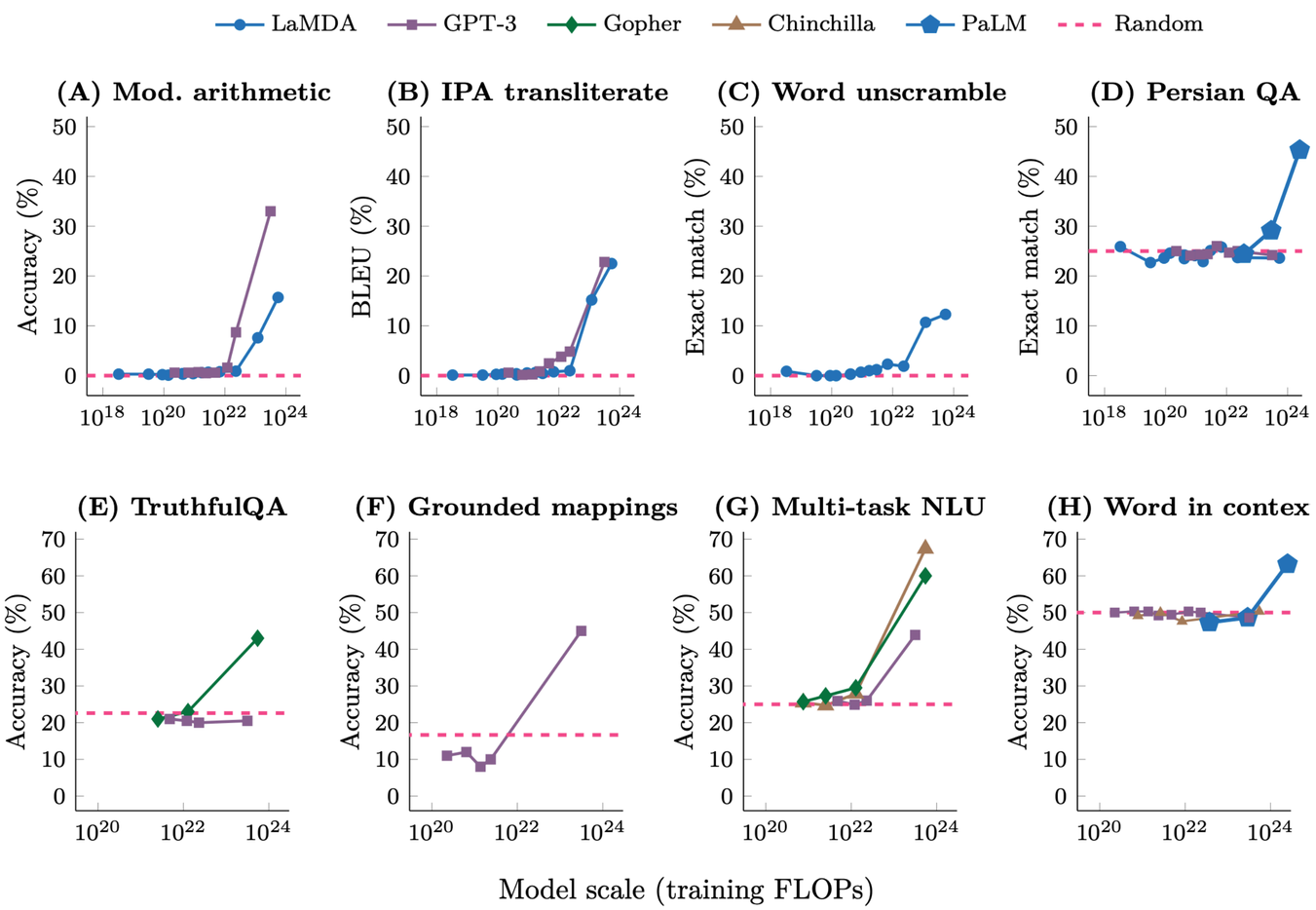


Disability Aid

The Scaling Law and Emergent Abilities

Scaling up improves data efficiency and unlocks emergent abilities



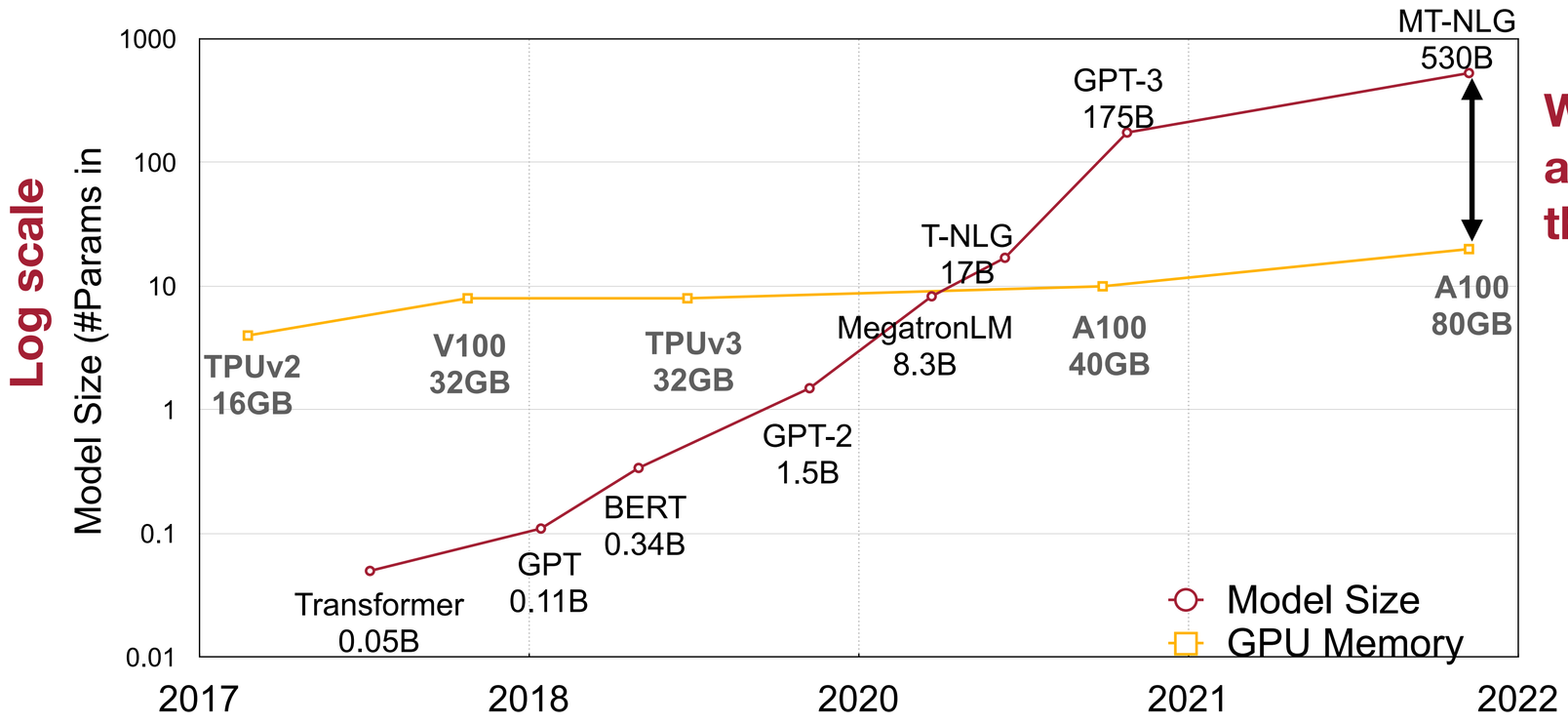


Scaling Laws for Neural Language Models (Kaplan et al., 2020)
Emergent Abilities of Large Language Models (Wei et al., 2022)

LLM Models Outgrow Hardware Capacity

Reducing LLM serving cost is essential

- LLM sizes and computation are increasing exponentially.
- Domain-specific accelerator alone is not enough; we need efficient deployment algorithms to bridge the gap.

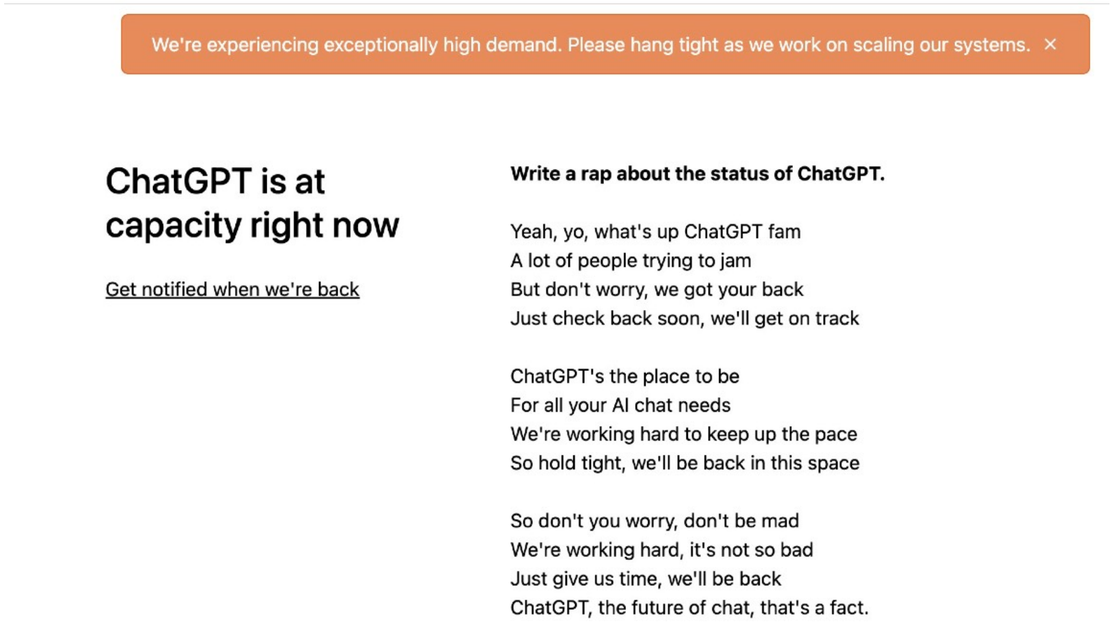


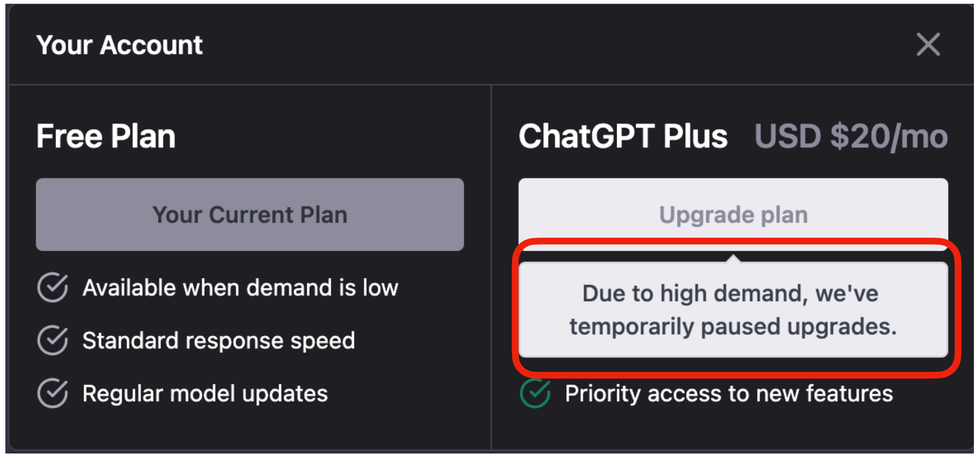
We need efficient algorithms to bridge the gap.

LLM Models Outgrow Hardware Capacity

Reducing LLM serving cost is essential

- LLM sizes and computation are increasing exponentially.
- Domain-specific accelerator alone is not enough; we need model compression techniques to bridge the gap.

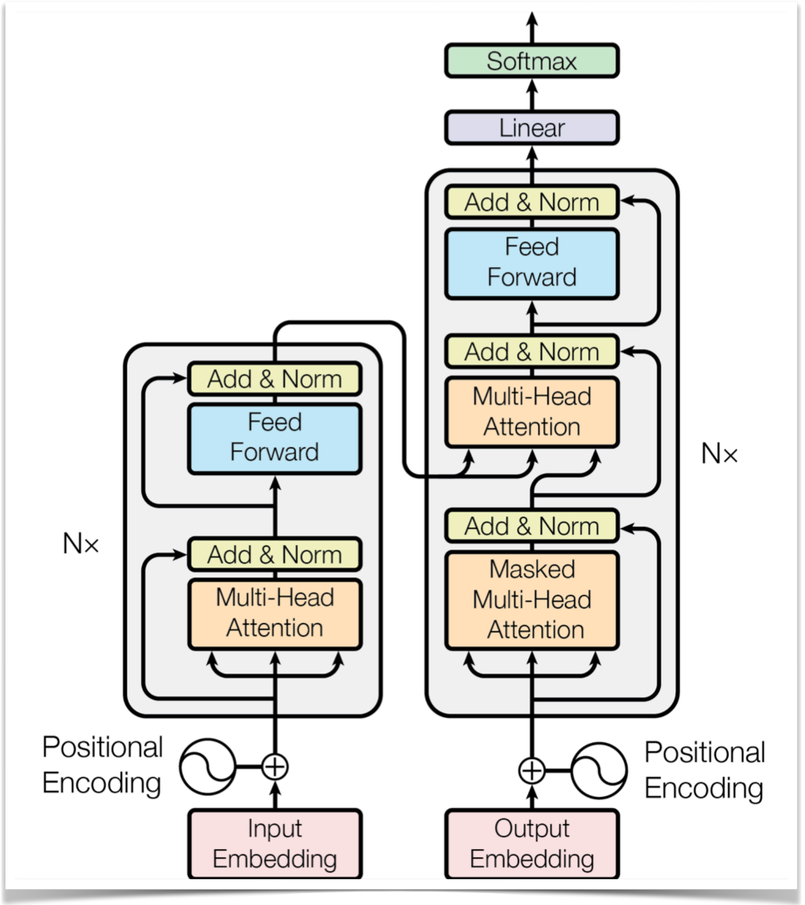




Lecture Plan

Today, we will cover:

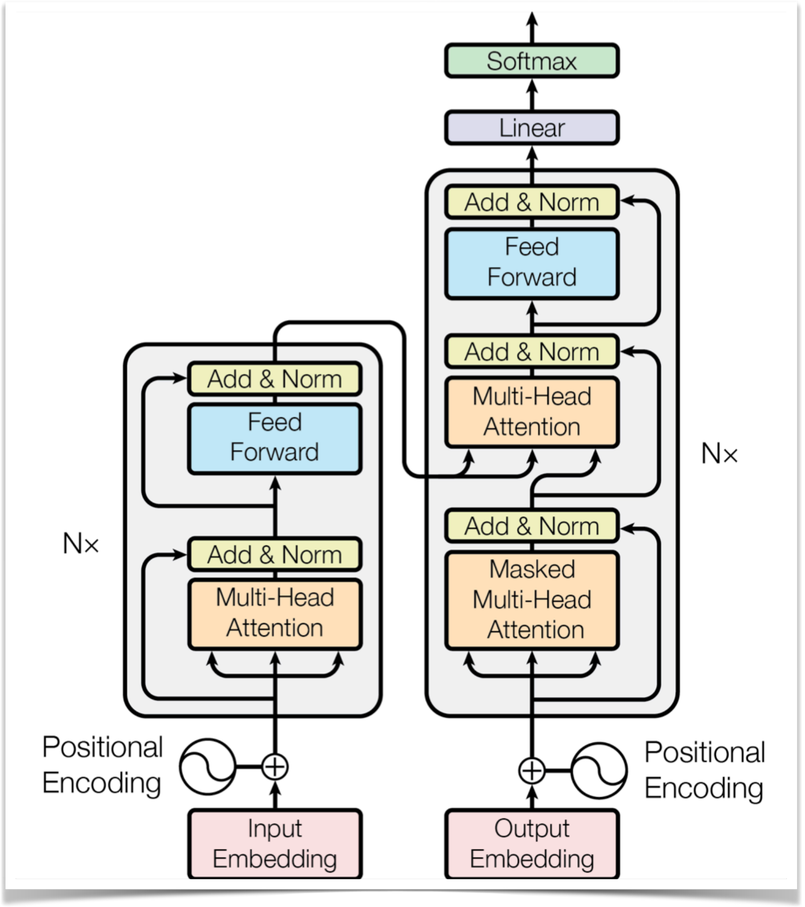
1. **Linear Quantization Basics**
2. **Weight-Activation Quantization: SmoothQuant**
3. **Weight-Only Quantization: AWQ and TinyChat**
4. **Pushing Further: QServe (W4A8KV4)**



Lecture Plan

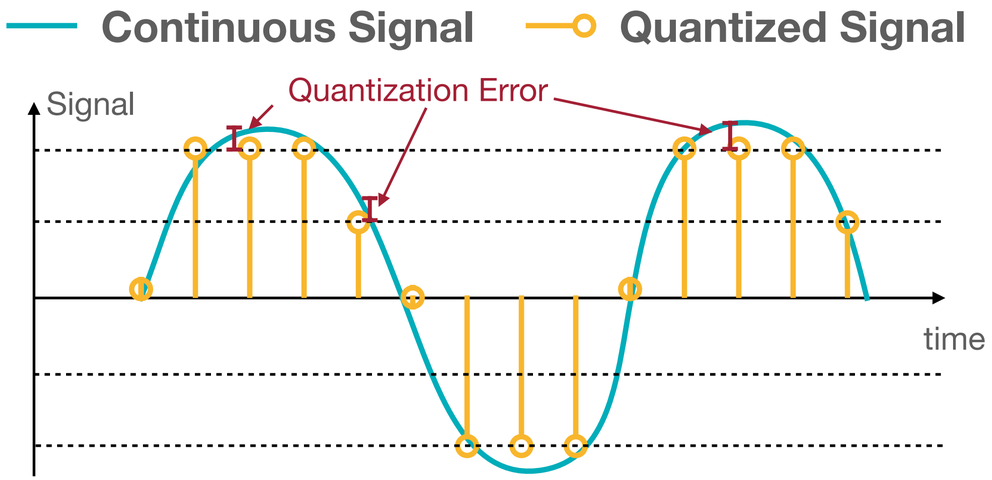
Today, we will cover:

1. **Linear Quantization Basics**
2. Weight-Activation Quantization: SmoothQuant
3. Weight-Only Quantization: AWQ and TinyChat
4. Pushing Further: QServe (W4A8KV4)



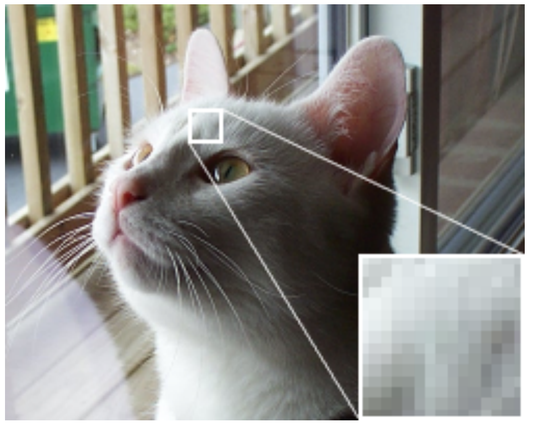
What is Quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.

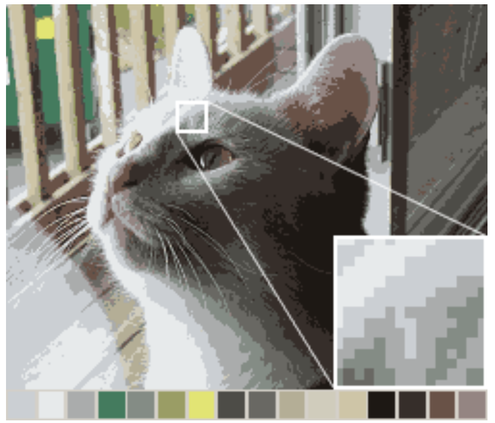


The difference between an input value and its quantized value is referred to as quantization error.

Original Image



16-Color Image



Images are in the public domain.

"Palettization"

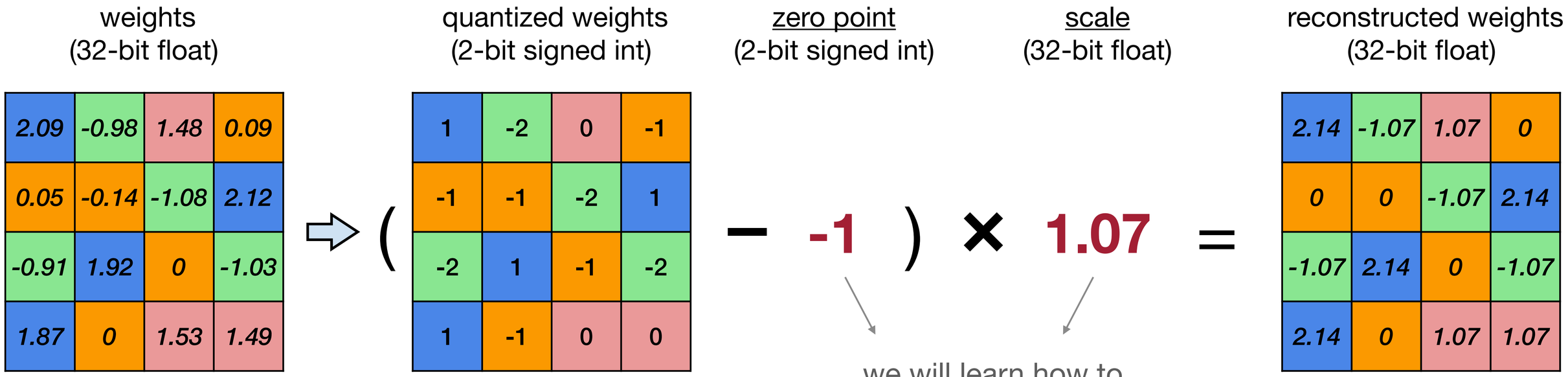
Quantization [Wikipedia]

What is Linear Quantization?

weights
(32-bit float)

2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

What is Linear Quantization?



we will learn how to determine these parameters

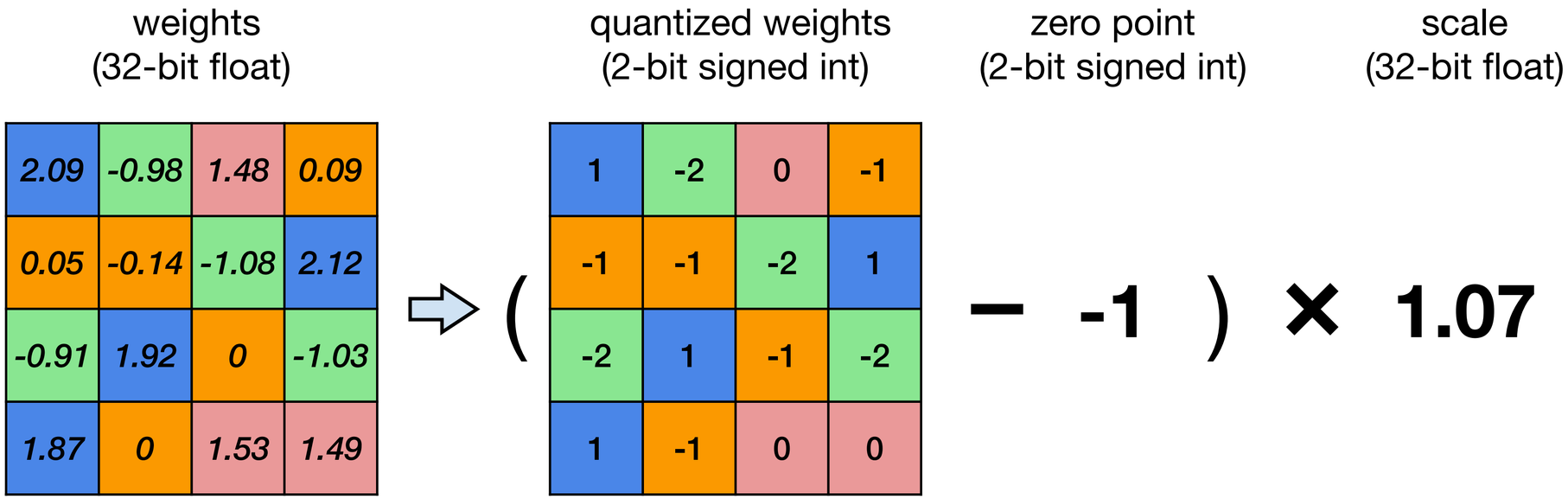
Binary	Decimal
01	1
00	0
11	-1
10	-2

quantization error

-0.05	*0.09*	*0.41*	*0.09*
0.05	*-0.14*	*-0.01*	*-0.02*
0.16	*-0.22*	*0*	*0.04*
-0.27	*0*	*0.46*	*0.42*

Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



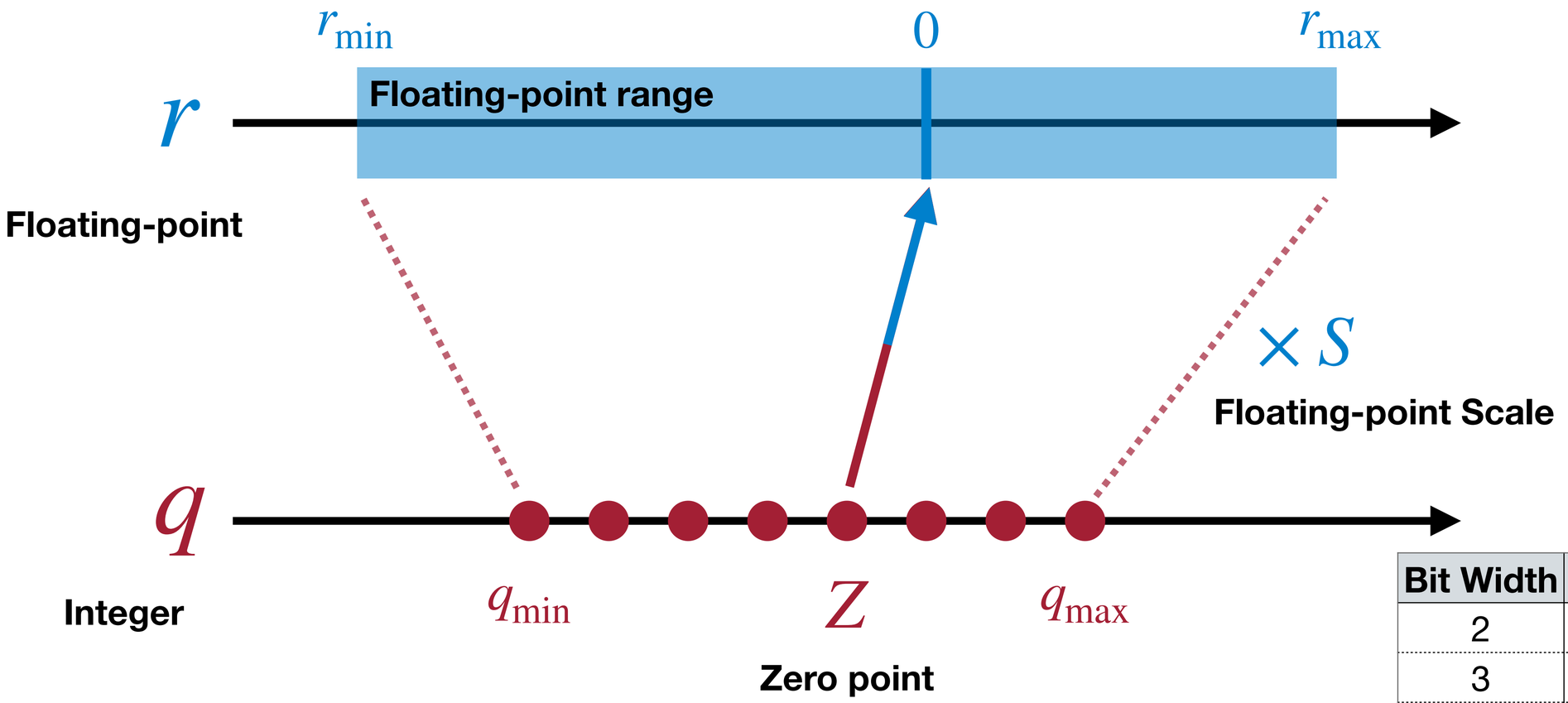
- quantization parameter
- allow real number *r=0* be exactly representable by a quantized integer Z

- quantization parameter

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$

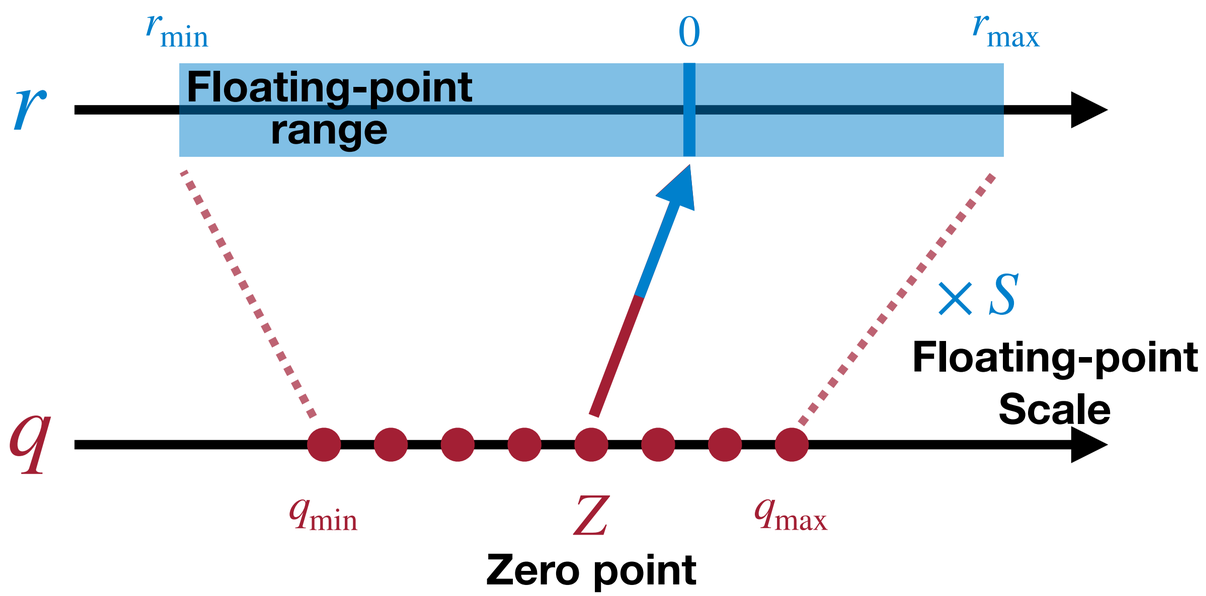


Bit Width	q_{min}	q_{max}
2	-2	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1}-1$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Scale of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{max} = S\left(q_{max} - Z\right)$$

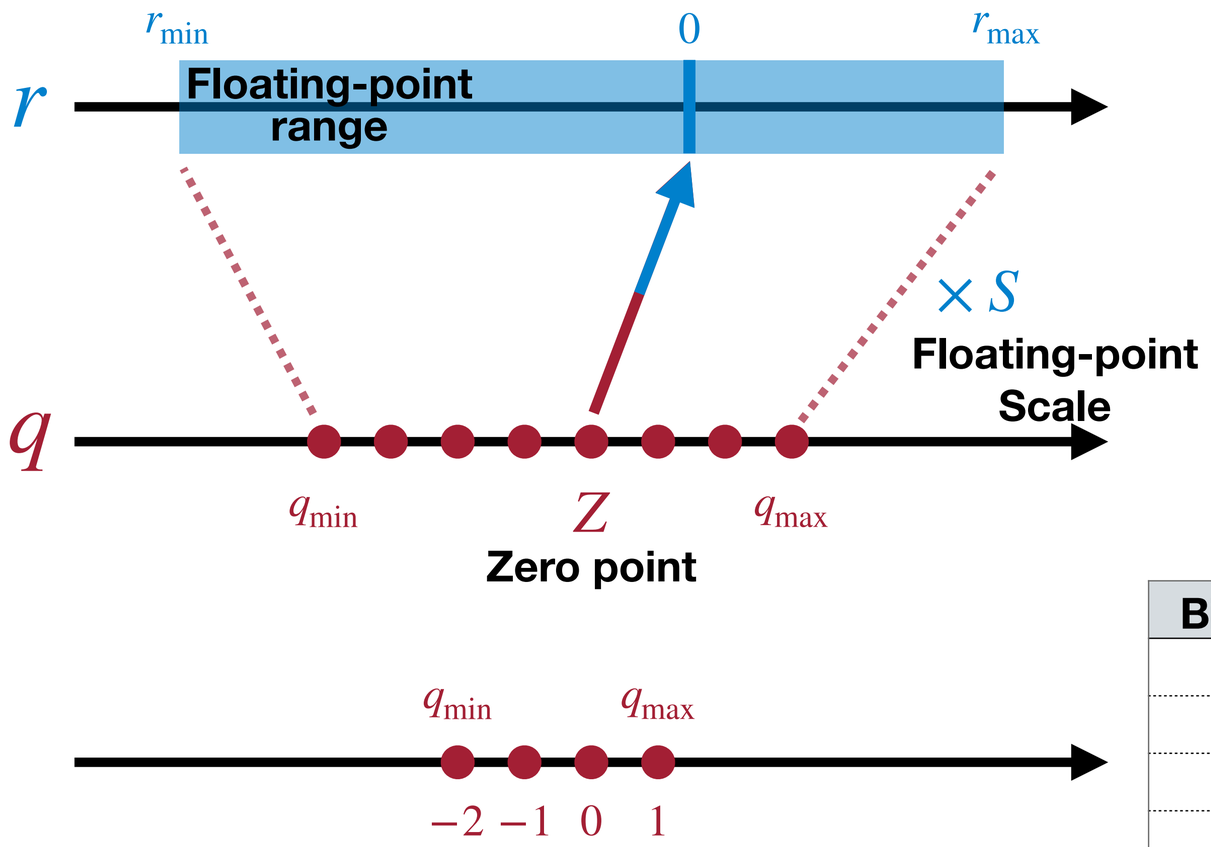
$$r_{min} = S\left(q_{min} - Z\right)$$

$$r_{max} - r_{min} = S\left(q_{max} - q_{min}\right)$$

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

Scale of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

Binary	Decimal
01	1
00	0
11	-1
10	-2

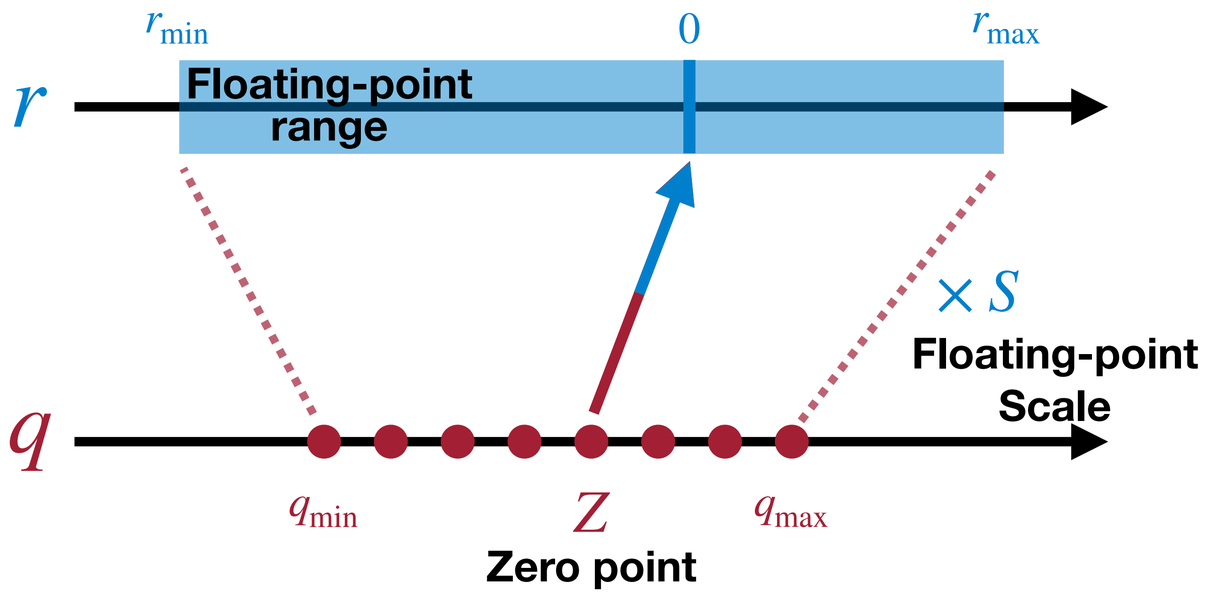
$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

$$= \frac{2.12 - (-1.08)}{1 - (-2)}$$

$$= 1.07$$

Zero Point of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{min} = S\left(q_{min} - Z\right)$$

$$\downarrow$$

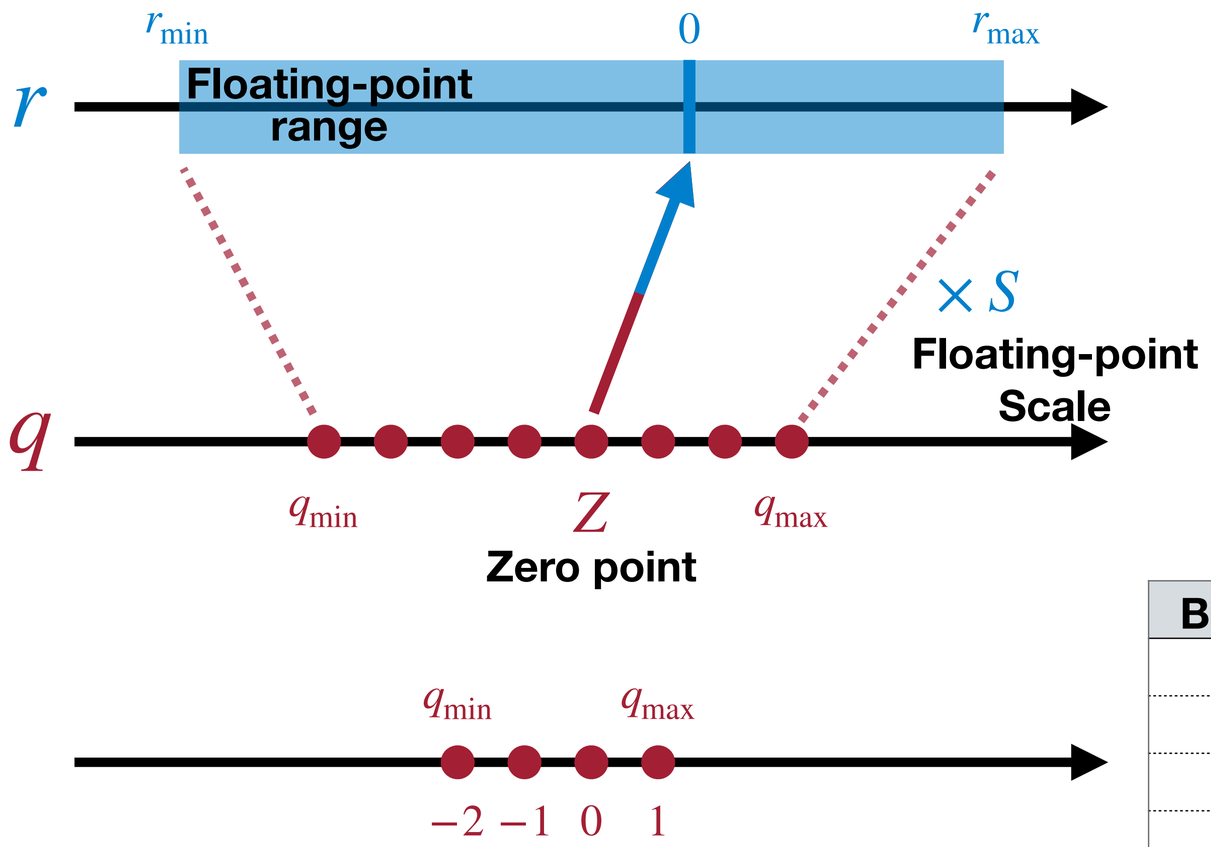
$$Z = q_{min} - \frac{r_{min}}{S}$$

$$\downarrow$$

$$Z = \text{round}\left(q_{min} - \frac{r_{min}}{S}\right)$$

Zero Point of Linear Quantization

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$



2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

Binary	Decimal
01	1
00	0
11	-1
10	-2

$$Z = q_{min} - \frac{r_{min}}{S}$$

$$= \text{round}(-2 - \frac{-1.08}{1.07})$$

$$= -1$$

Linear Quantized Matrix Multiplication

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication.

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$$S_{\mathbf{Y}}\left(\mathbf{q}_{\mathbf{Y}} - Z_{\mathbf{Y}}\right) = S_{\mathbf{W}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right) \cdot S_{\mathbf{X}}\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right)$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}\left(\mathbf{q}_{\mathbf{W}} - Z_{\mathbf{W}}\right)\left(\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\right) + Z_{\mathbf{Y}}$$

$$\mathbf{q}_{\mathbf{Y}} = \frac{S_{\mathbf{W}} S_{\mathbf{X}}}{S_{\mathbf{Y}}}\left(\mathbf{q}_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{W}}\mathbf{q}_{\mathbf{X}} - Z_{\mathbf{X}}\mathbf{q}_{\mathbf{W}} + Z_{\mathbf{W}}Z_{\mathbf{X}}\right) + Z_{\mathbf{Y}}$$

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Linear Quantized Matrix Multiplication

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication.

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$$\mathbf{q}_\mathbf{Y} = \frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}} \left(\mathbf{q}_\mathbf{W}\mathbf{q}_\mathbf{X} - Z_\mathbf{W}\mathbf{q}_\mathbf{X} - Z_\mathbf{X}\mathbf{q}_\mathbf{W} + Z_\mathbf{W}Z_\mathbf{X} \right) + Z_\mathbf{Y}$$

Precompute

***N*-bit Integer Multiplication**
32-bit Integer Addition/Subtraction

***N*-bit Integer Addition**

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Linear Quantized Matrix Multiplication

Linear Quantization is an affine mapping of integers to real numbers $r = S(q - Z)$

- Consider the following matrix multiplication.

$$\mathbf{Y} = \mathbf{W}\mathbf{X}$$

$$\mathbf{q}_\mathbf{Y} = \frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}} \left(\mathbf{q}_\mathbf{W}\mathbf{q}_\mathbf{X} - Z_\mathbf{W}\mathbf{q}_\mathbf{X} - Z_\mathbf{X}\mathbf{q}_\mathbf{W} + Z_\mathbf{W}Z_\mathbf{X} \right) + Z_\mathbf{Y}$$

- Empirically, the scale $\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}}$ is always in the interval (0, 1).

Fixed-point Multiplication

$$\frac{S_\mathbf{W} S_\mathbf{X}}{S_\mathbf{Y}} = 2^{-n} M_0, \quad \text{where} \quad M_0 \in [0.5, 1)$$

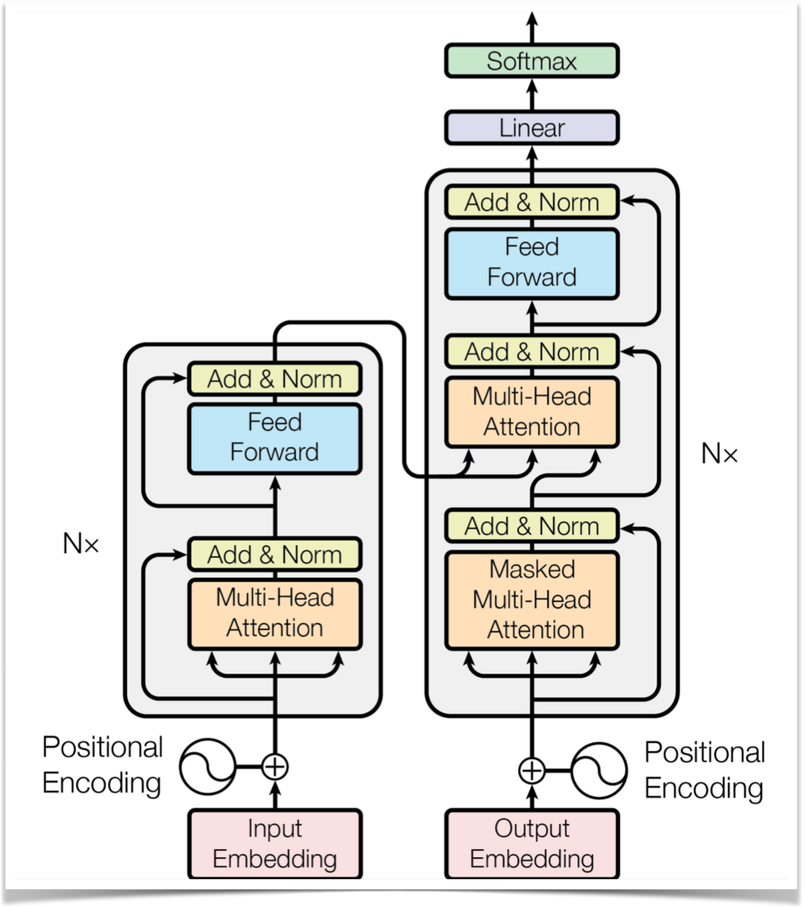
Bit Shift

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob *et al.*, CVPR 2018]

Lecture Plan

Today, we will cover:

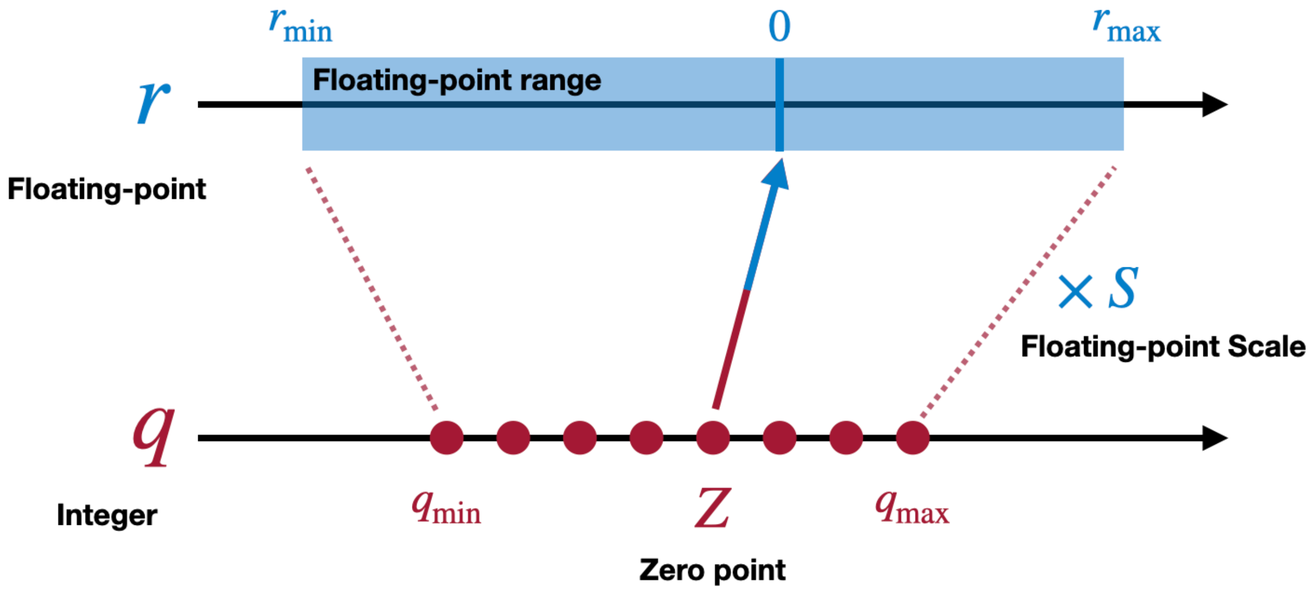
1. Linear Quantization Basics
2. **Weight-Activation Quantization: SmoothQuant**
3. Weight-Only Quantization: AWQ and TinyChat
4. Further Practice: QServe (W4A8KV4)



Quantization Can Reduce Deployment Costs

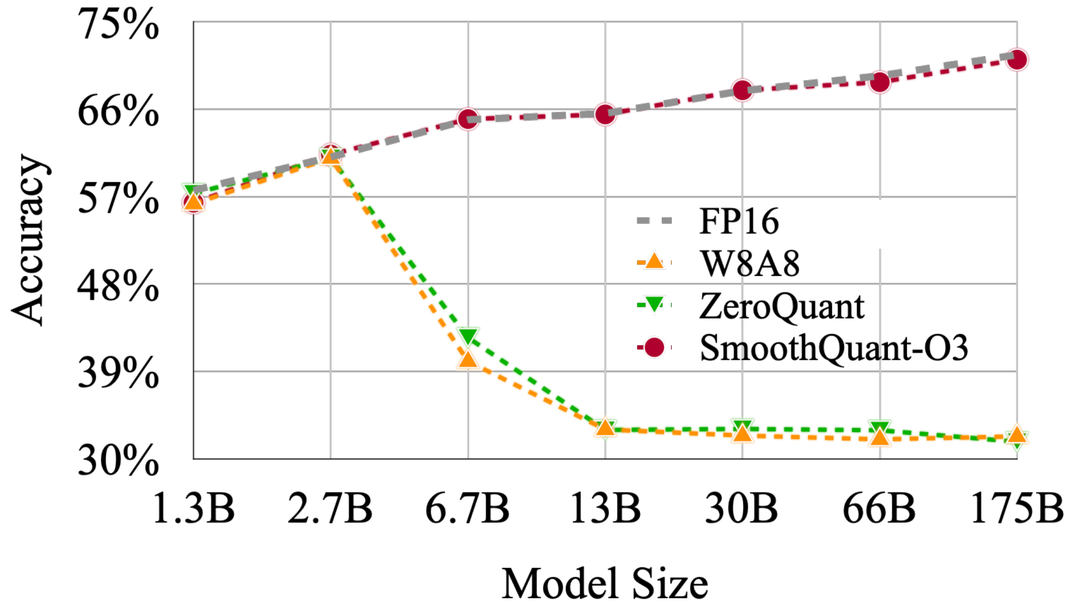
Quantization lowers the bit-width and improves efficiency

- Serving a 175B GPT-3 model at least requires:
  - FP16: 350GB memory ➡️ 5 x 80GB A100 GPUs
  - **INT8:** 175GB memory ➡️ **3 x 80GB A100 GPUs**



Naive Quantization Methods are Inaccurate

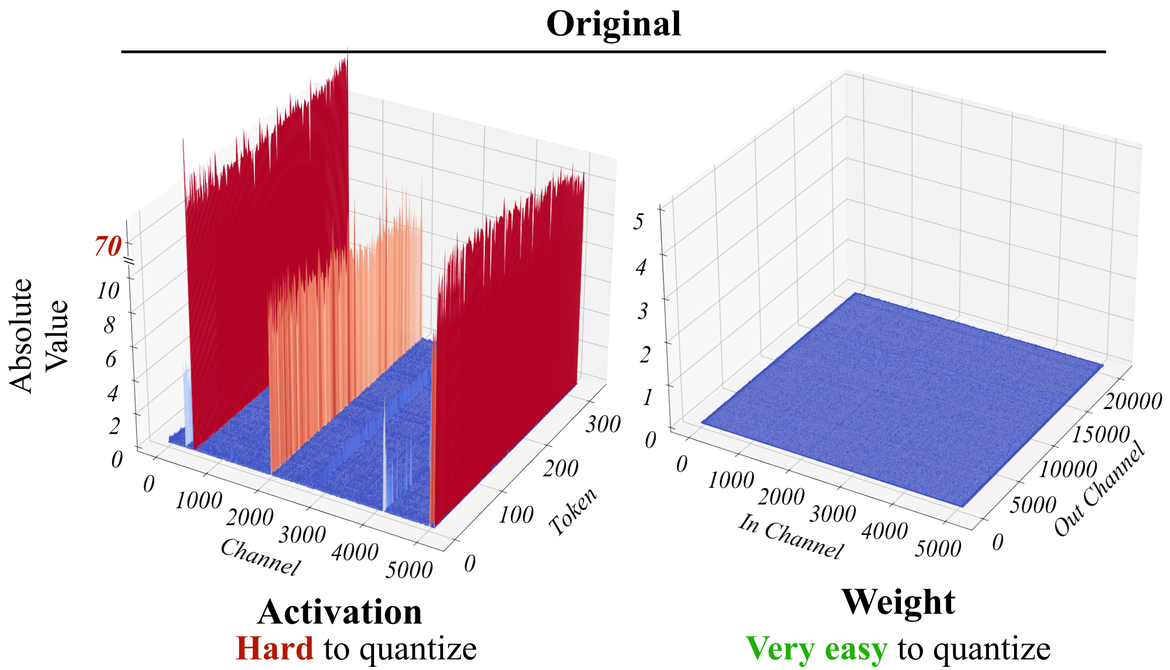
Activation outliers destroy quantized performance



- W8A8 quantization has been an industrial standard for CNNs, but not LLM. Why?
- Systematic outliers emerge in **activations** when we scale up LLMs beyond 6.7B. Traditional CNN quantization methods will destroy the accuracy.

Understanding the Quantization Difficulty of LLMs

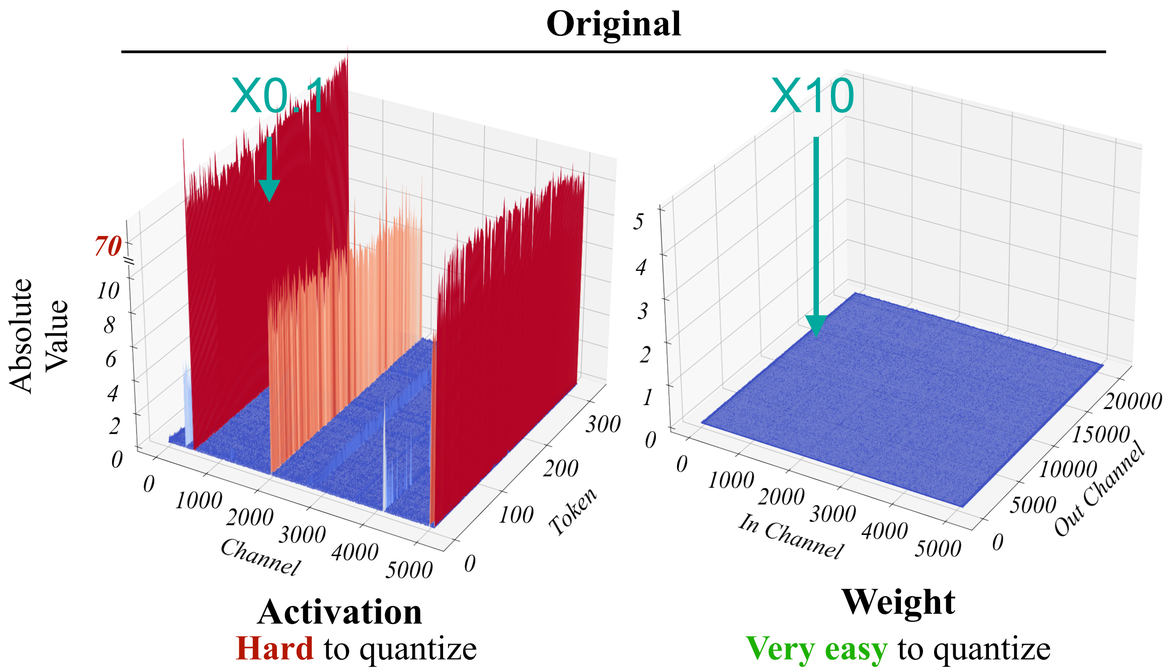
Smoothing activation to reduce quantization error



- Weights are easy to quantize, but activation is hard due to outliers

Understanding the Quantization Difficulty of LLMs

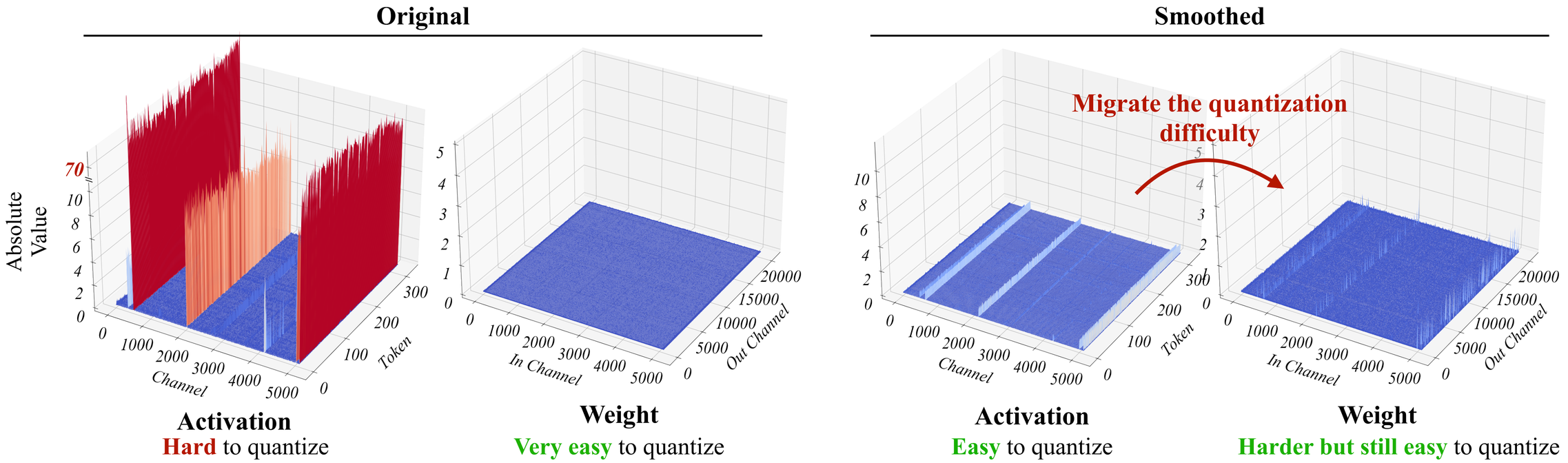
Smoothing activation to reduce quantization error



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

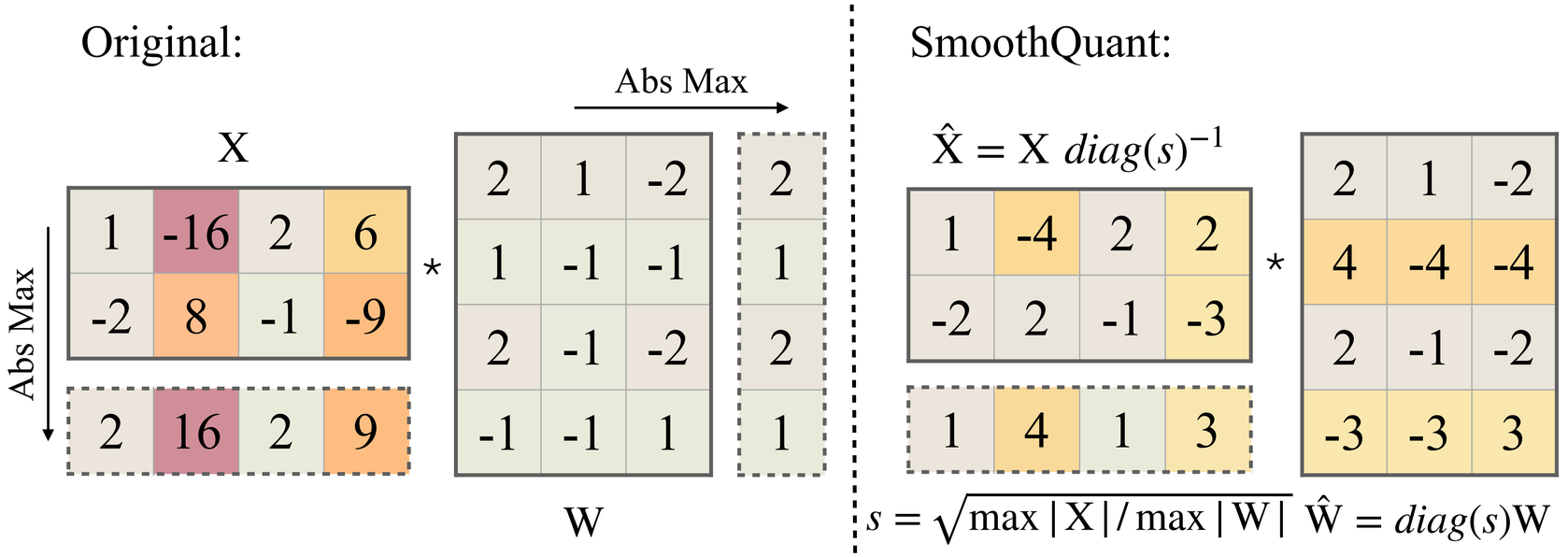
Understanding the Quantization Difficulty of LLMs

Smoothing activation to reduce quantization error



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels
- Migrate the quantization difficulty from activation to weights, so both are easy to quantize

Activation Smoothing



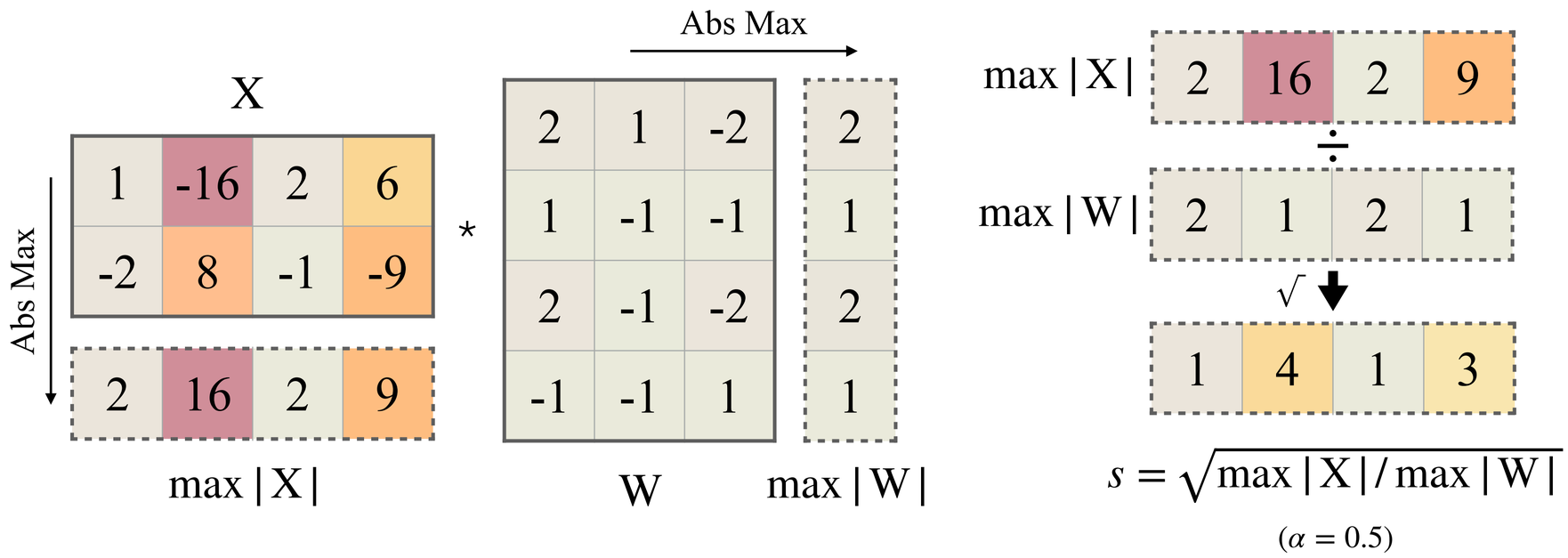
$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha},\ j = 1,2,\ldots,C_i$$

$$\mathbf{Y} = (\mathbf{X}diag(\mathbf{s})^{-1}) \cdot (diag(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

α: Migration Strength

Activation Smoothing

1.Calibration Stage (Offline):

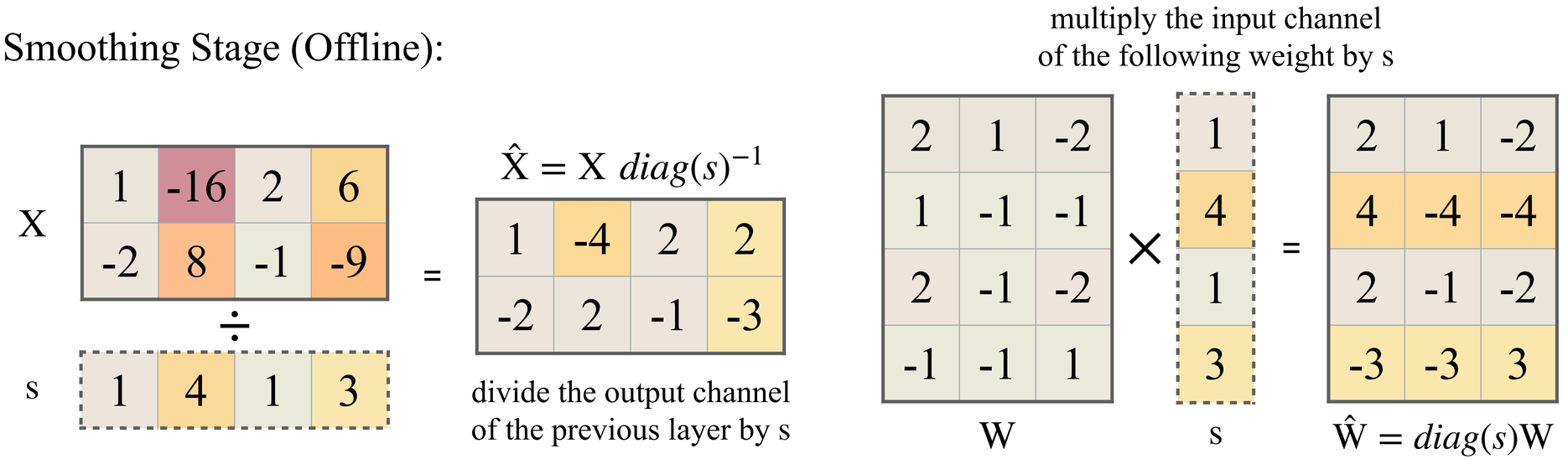


$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha},\ j = 1,2,\ldots,C_i$$

α: Migration Strength

Activation Smoothing

2. Smoothing Stage (Offline):



$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha},\ j = 1,2,\ldots,C_i$$

$$\mathbf{Y} = (\mathbf{X}diag(\mathbf{s})^{-1}) \cdot (diag(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

α: Migration Strength

Activation Smoothing

3. Inference (deployed model):

$\hat{X}$

1	-4	2	2
-2	2	-1	-3

At runtime, the activations are smooth and easy to quantize

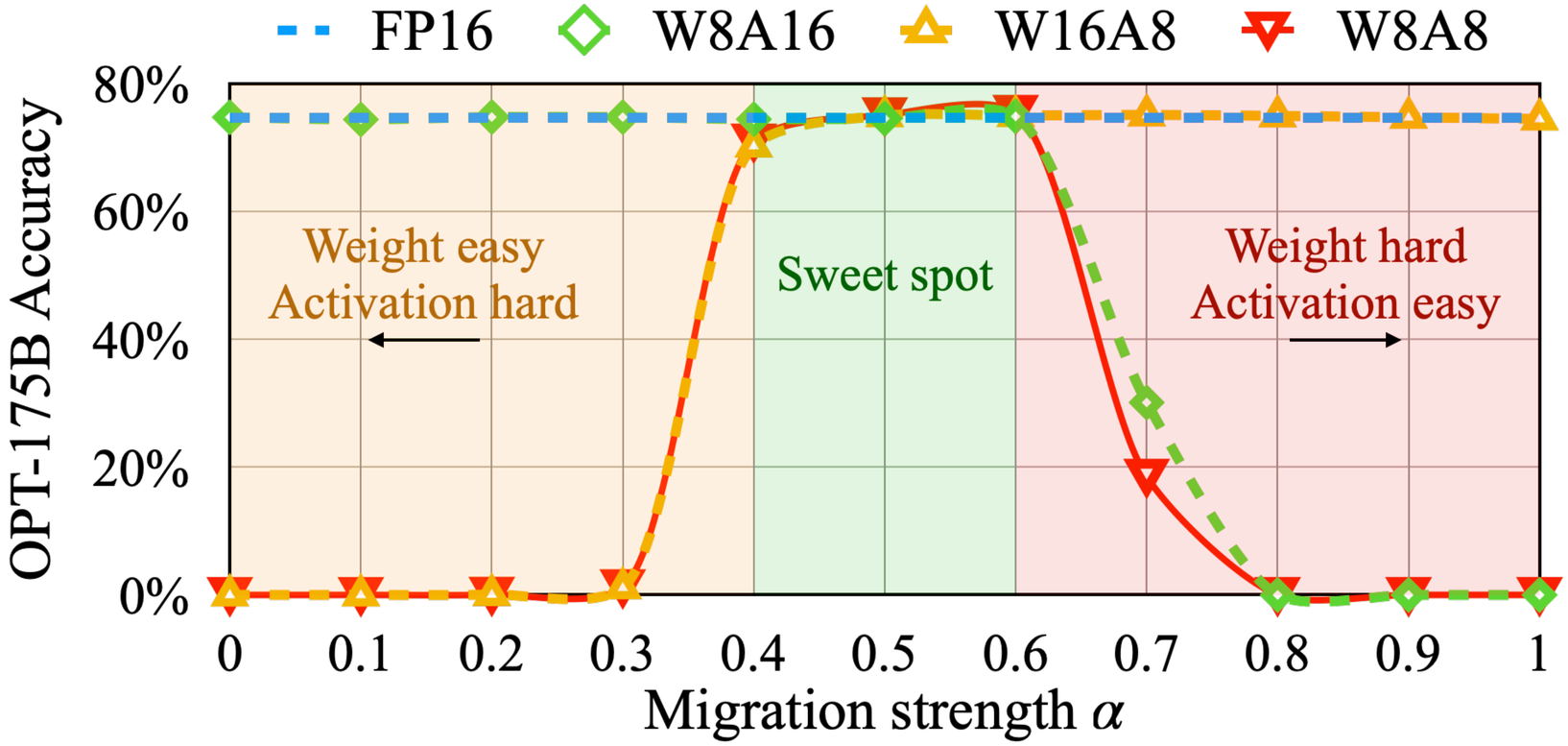
*

2	1	-2
4	-4	-4
2	-1	-2
-3	-3	3

$\hat{W}$

$$\mathbf{Y} = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

Ablation Study on the Migration Strength α



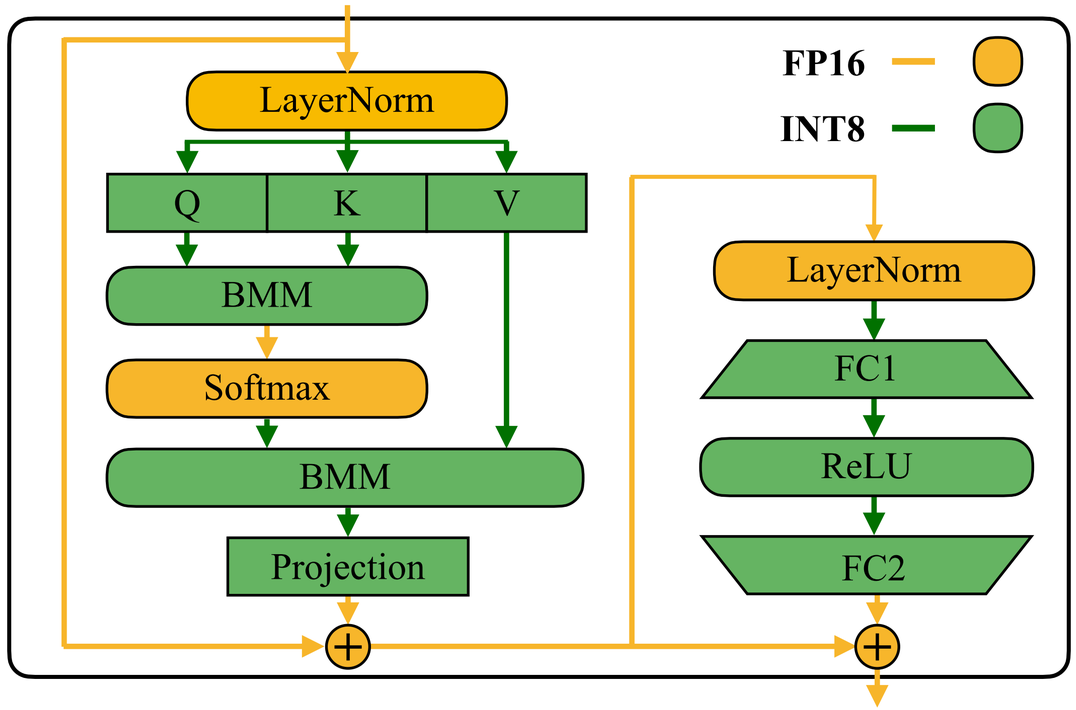
$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha},\ j = 1,2,\ldots,C_i \qquad \mathbf{Y} = (\mathbf{X}diag(\mathbf{s})^{-1}) \cdot (diag(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

- Migration strength α controls the amount of quantization difficulty migrated from activations to weights.
- A suitable migration strength α (sweet spot) makes both activations and weights easy to quantize.
- If the α is too large, weights will be hard to quantize; if too small, activations will be hard to quantize.

System Implementation

Efficient System Implementation

- We integrate SmoothQuant into FasterTransformer
- All compute-intensive operators (Linear, BMM) are quantized

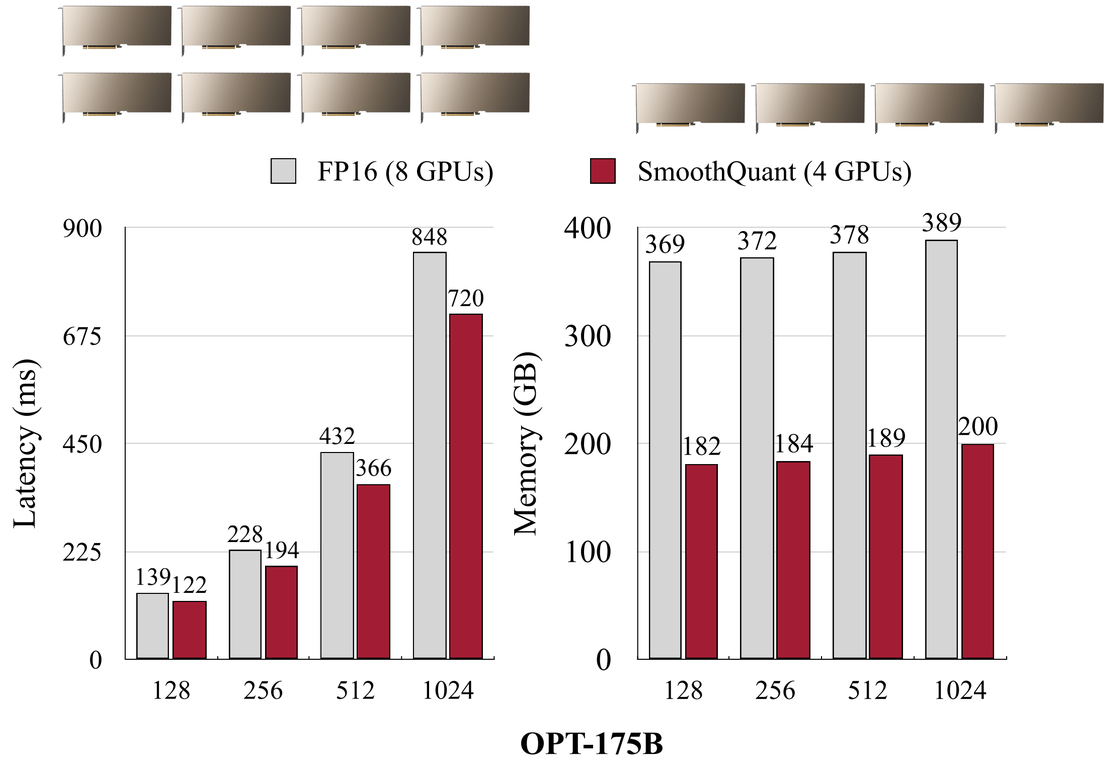


SmoothQuant is Accurate and Efficient

- SmoothQuant well maintains the accuracy without fine-tuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.

LAMBADA Accuracy

	OPT-175B	BLOOM-176B	GLM-130B
FP16	71.6%	68.2%	73.8%
SmoothQuant	71.2%	68.3%	73.7%



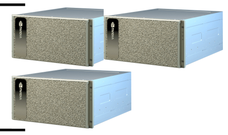
Scaling Up: 530B Model Within a Single Node

MT-NLG 530B Accuracy

	LAMBADA	HellaSwag	PIQA	WinoGrande	Average
FP16	76.6%	62.1%	81.0%	72.9%	73.1%
INT8	77.2%	60.4%	80.7%	74.1%	73.1%

MT-NLG 530B Efficiency

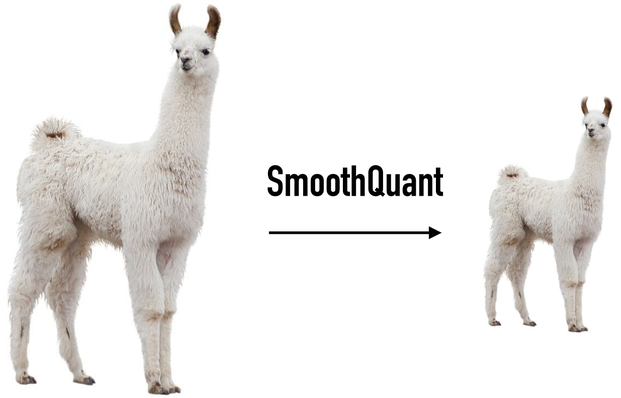
SeqLen	Prec.	#GPUs	Latency	Memory
512	FP16	16	838ms	1068GB
	INT8	8	839ms	545GB
1024	FP16	16	1707ms	1095GB
	INT8	8	1689ms	570GB



- SmoothQuant can accurately quantize MT-NLG 530B model and reduce the serving GPU numbers by half at a similar latency, which allows serving the 530B model within a single node.

SmoothQuant

Advancing new efficient open model LLaMA



- **LLaMA** (and its successors like Alpaca) are popular open-source LLMs, which introduced SwishGLU, making activation quantization even harder
- SmoothQuant can losslessly quantize LLaMA families, further lowering the hardware barrier

Wikitext↓	LLaMA 7B	LLaMA 13B	LLaMA 30B	LLaMA 65B
FP16	11.51	10.05	7.53	6.17
SmoothQuant	11.56	10.08	7.56	6.20

Industry & Community Impact

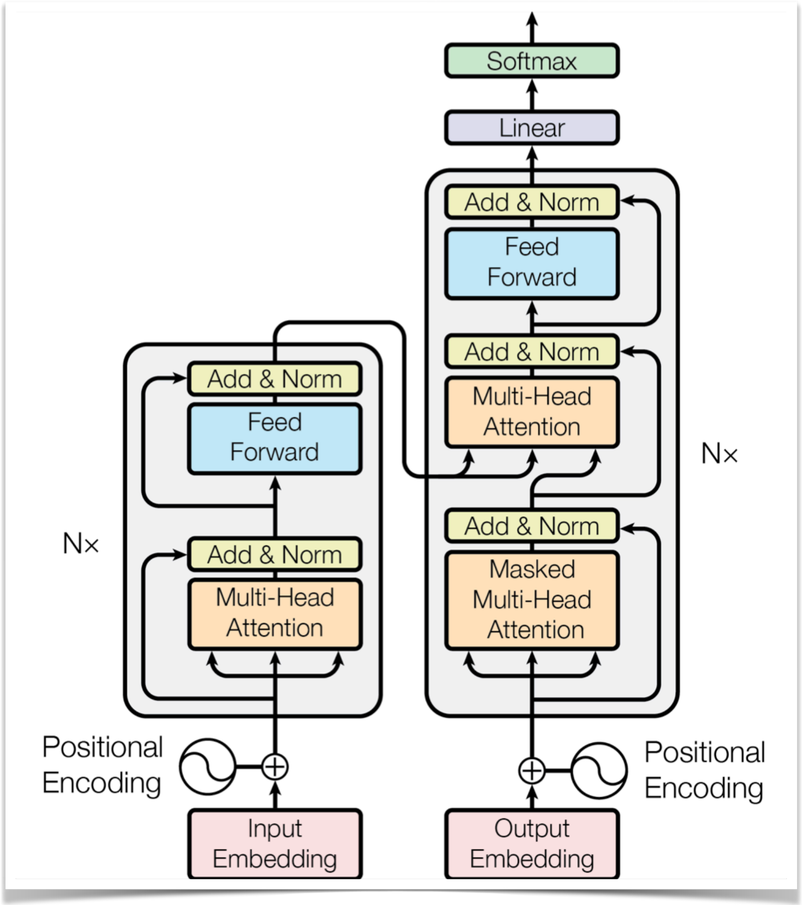
SmoothQuant is widely adopted by industry

- NVIDIA FasterTransformer
- NVIDIA TRT-LLM
- MLPerf 8-bit: closed the accuracy gap.
- Intel Neural Compressor / Q8-Chat on Xeon
- Meta/Microsoft/Amazon/HuggingFace …

Lecture Plan

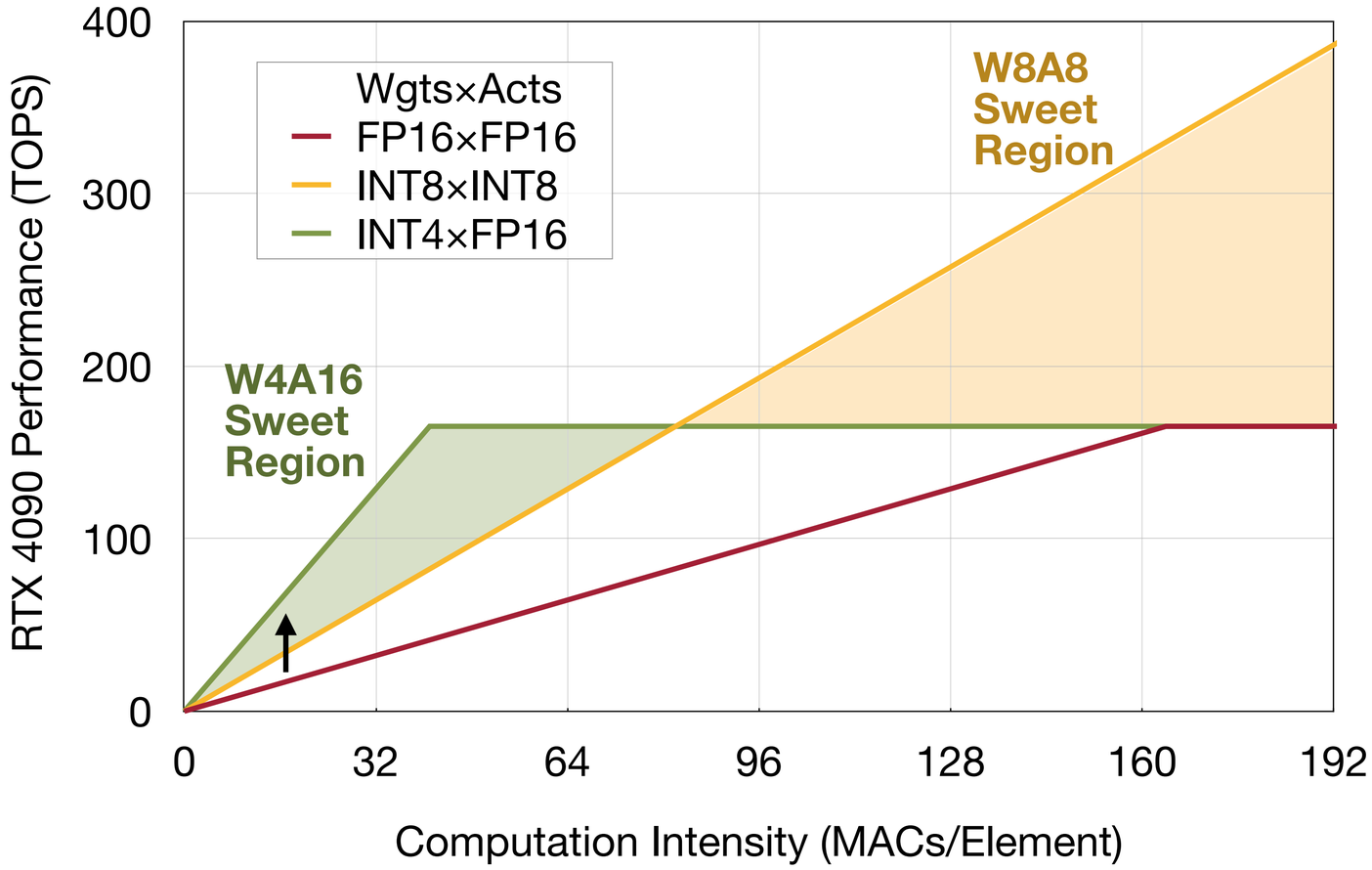
Today, we will cover:

1. Linear Quantization Basics
2. Weight-Activation Quantization: SmoothQuant
3. **Weight-Only Quantization: AWQ and TinyChat**
4. Pushing Further: QServe (W4A8KV4)



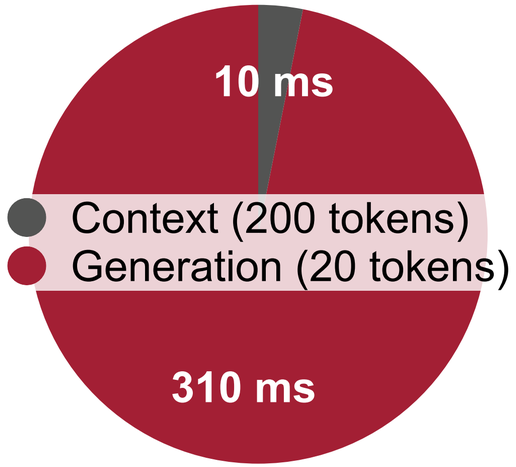
Is W8A8 quantization enough?

Edge LLMs are memory bound and requires low-bit weight quantization

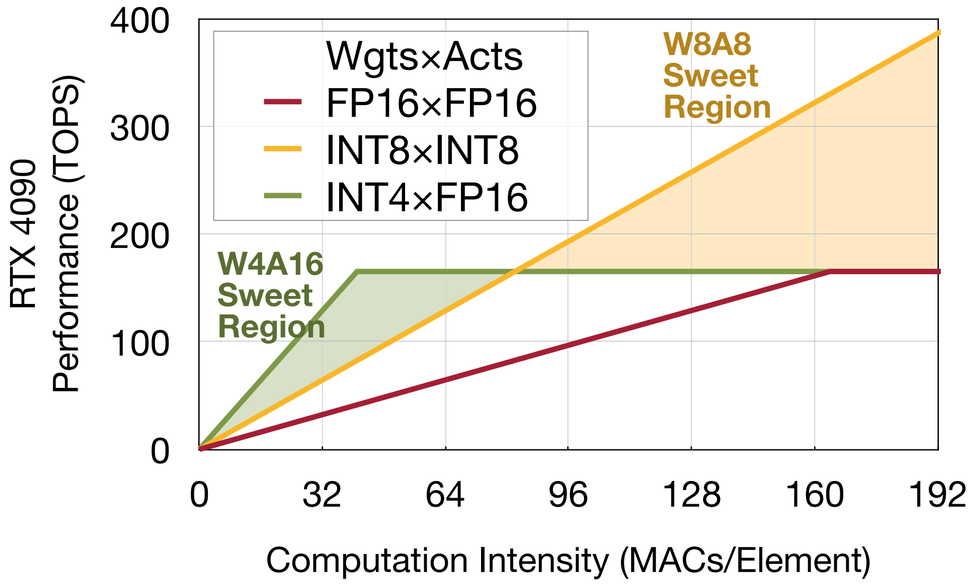


Low-bit weight quantization brings speedup

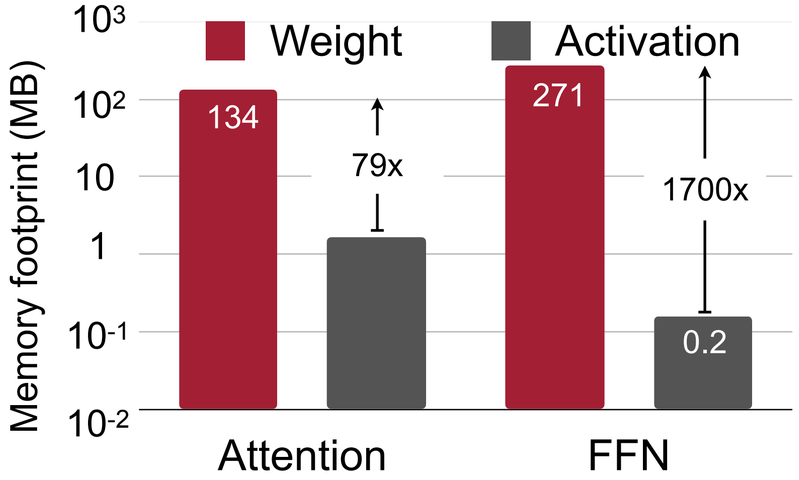
Weight loading is the efficiency bottleneck for edge LLM inference



(a) Generation stage is slower



(b) Generation stage is bounded by memory bandwidth



(c) Weight loading is more expensive

AWQ: Activation-aware Weight Quantization

Targeting group-wise low-bit weight-only quantization (W4A16)

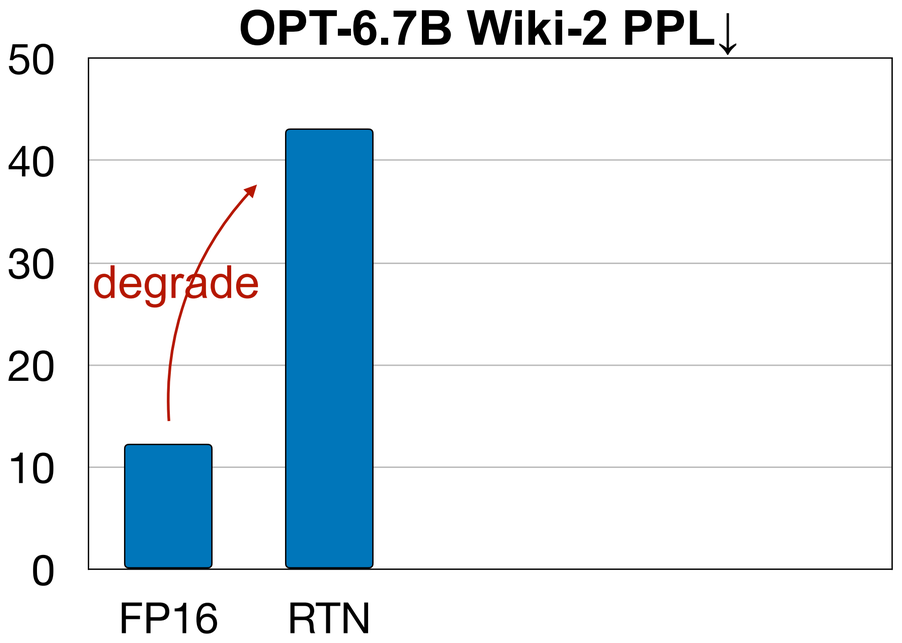
$\mathbf{W}_{\text{FP16}}$

+1.2	−0.2	−2.4	−3.4
−2.5	−3.5	+1.9	+1.4
−0.9	+1.6	−2.5	−1.9
−3.5	+1.5	+0.5	−0.1
+1.8	−1.6	−3.2	−3.4
+2.4	−3.5	−2.8	−3.9
+0.1	−3.8	+2.4	+3.4
+0.9	+3.3	−1.9	−2.3

RTN →

$Q(\mathbf{W})_{\text{INT3}}$

+1	+0	−2	−3
−3	−4	+2	+1
−1	+2	−3	−2
−4	+2	+1	+0
+2	−2	−3	−3
+2	−4	−3	−4
+0	−4	+2	+3
+1	+3	−2	−2



- Weight-only quantization reduces the memory requirement, and accelerates token generation by alleviating the memory bottleneck.
- Group-wise/block-wise quantization (e.g., 64/128/256) offers a better accuracy-model size trade-off.
- But there is still a performance gap with round-to-nearest (RTN) quantization (INT3-g128)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

AWQ: Activation-aware Weight Quantization

Observation: Weights are not equally important; 1% salient weights

$\mathbf{W}_{\text{FP16}}$

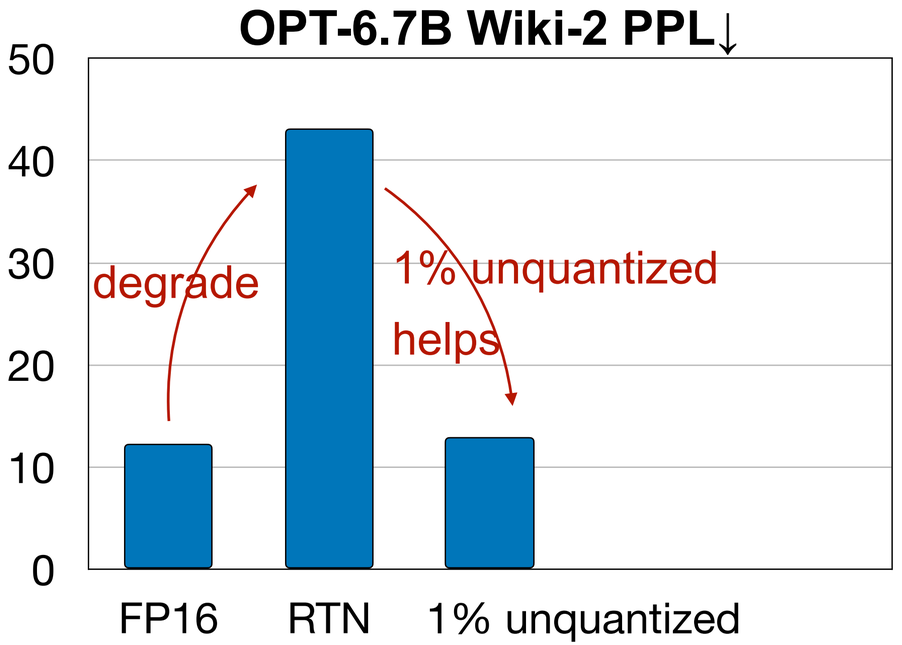
+1.2	−0.2	−2.4	−3.4
−2.5	−3.5	+1.9	+1.4
−0.9	+1.6	−2.5	−1.9
−3.5	+1.5	+0.5	−0.1
+1.8	−1.6	−3.2	−3.4
+2.4	−3.5	−2.8	−3.9
+0.1	−3.8	+2.4	+3.4
+0.9	+3.3	−1.9	−2.3

RTN →

$Q(\mathbf{W})_{\text{MixPrec}}$

+1	+0	−2	−3
−2.5	−3.5	+1.9	+1.4
−1	+2	−3	−2
−4	+2	+1	+0
+2	−2	−3	−3
+2	−4	−3	−4
+0	−4	+2	+3
+1	+3	−2	−2

Unquantized channel



- We find that weights are not equally important, keeping **only 1%** of salient weight channels unquantized can greatly improve perplexity

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

AWQ: Activation-aware Weight Quantization

Observation: Weights are not equally important; 1% salient weights

$\mathbf{W}_{\text{FP16}}$

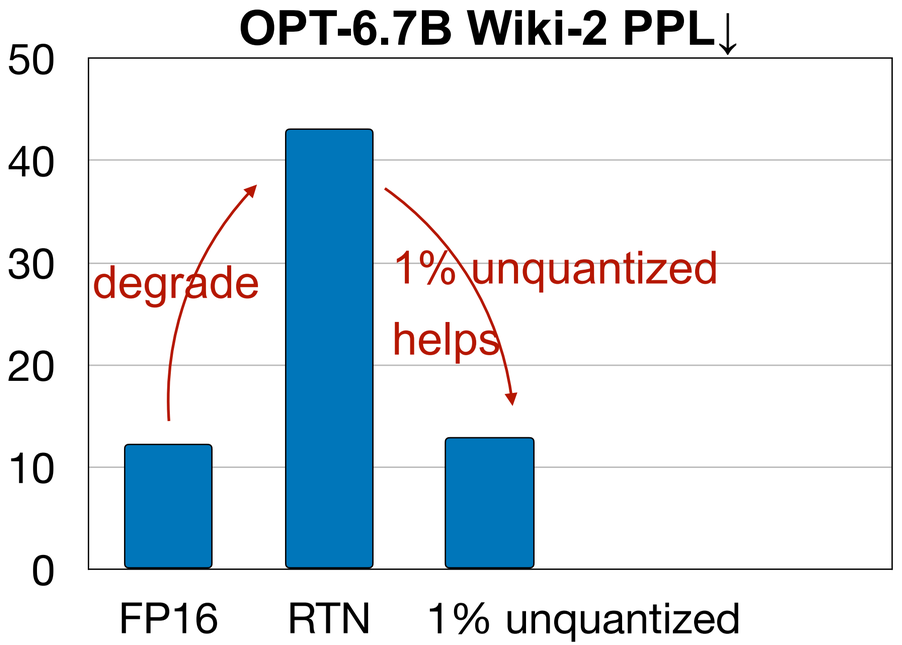
+1.2	−0.2	−2.4	−3.4
−2.5	−3.5	+1.9	+1.4
−0.9	+1.6	−2.5	−1.9
−3.5	+1.5	+0.5	−0.1
+1.8	−1.6	−3.2	−3.4
+2.4	−3.5	−2.8	−3.9
+0.1	−3.8	+2.4	+3.4
+0.9	+3.3	−1.9	−2.3

RTN →

$Q(\mathbf{W})_{\text{MixPrec}}$

+1	+0	−2	−3
−2.5	−3.5	+1.9	+1.4
−1	+2	−3	−2
−4	+2	+1	+0
+2	−2	−3	−3
+2	−4	−3	−4
+0	−4	+2	+3
+1	+3	−2	−2

Unquantized channel

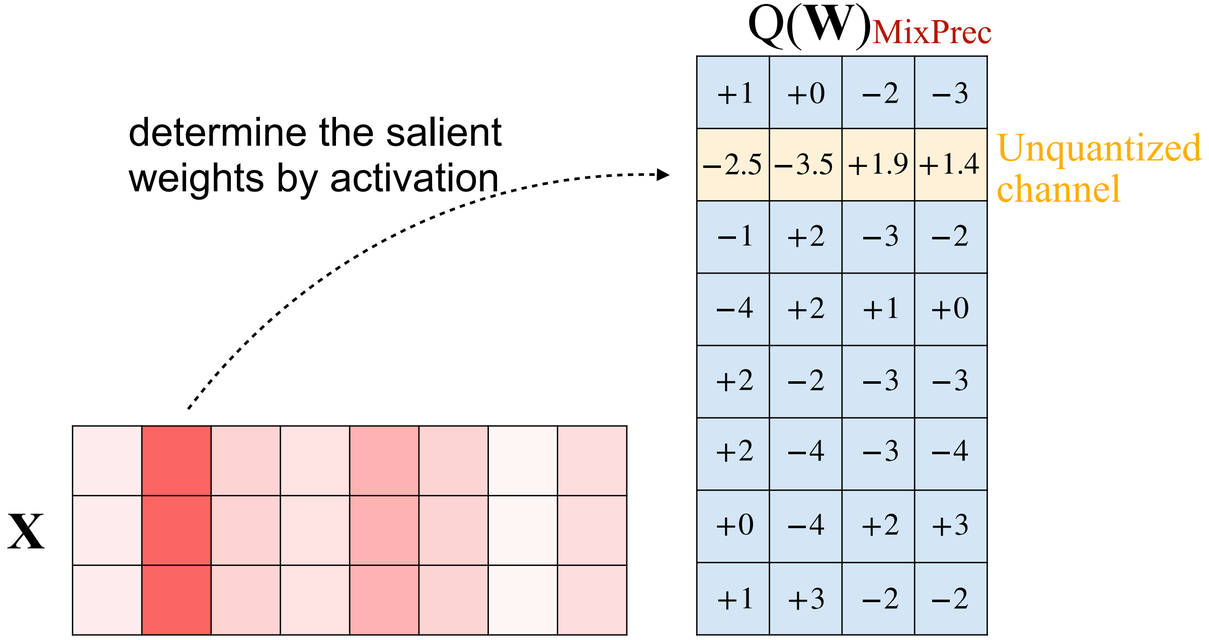


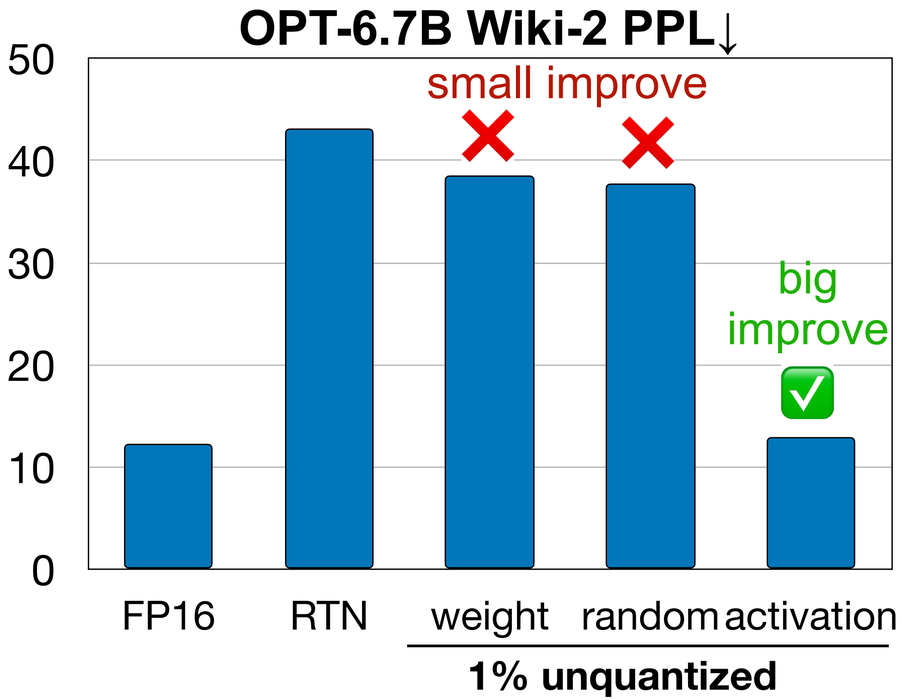
- We find that weights are not equally important, keeping **only 1%** of salient weight channels unquantized can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

AWQ for low-bit weight-only quantization

Salient weights are determined by activation distribution, not weight





- But how do we select salient channels? Should we select based on weight magnitude?
- No! We should look for **activation distribution, but not weight! (Activation has outliers!)**
- **However, is it really necessary to introduce mixed precision?**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

AWQ for low-bit weight-only quantization

Protecting salient weights by scaling (without mixed precision)

Quantization scaler, Absmax within the group

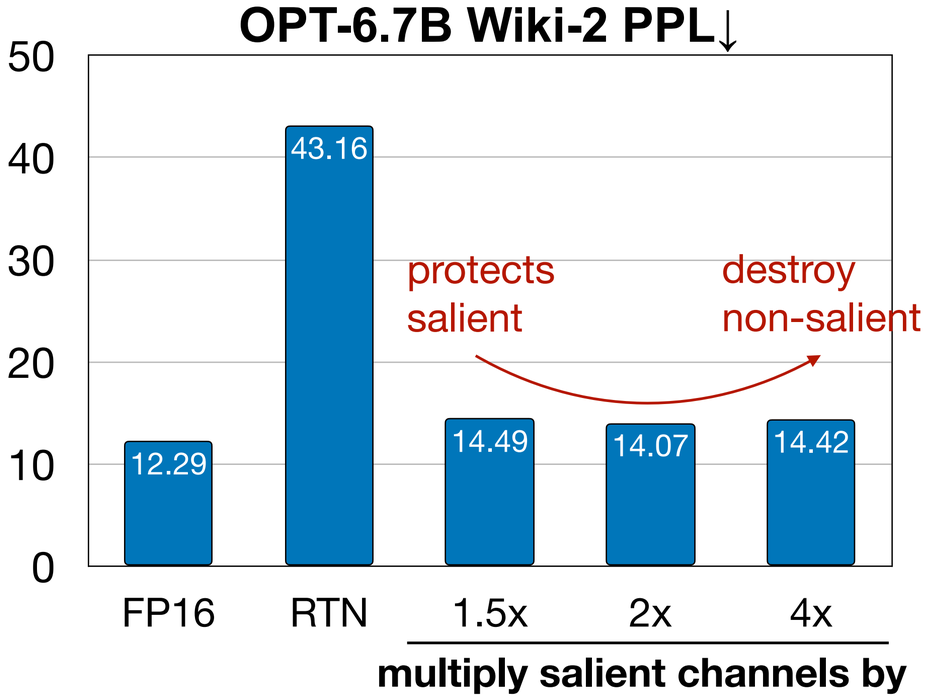
$$\text{Let} \quad Q(\mathbf{w}) = \Delta \cdot \text{round}(\frac{\mathbf{w}}{\Delta})$$

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

fuse to previous op

Quantize

$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



- We need to consider **activation-awareness** for salient channels.

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

Why AWQ reduces quantization error?

When scale s is not too large, quantization error is inversely proportional to s.

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

$$Q(w \cdot s) \cdot \frac{x}{s} = \Delta' \cdot \text{Round}(\frac{w \cdot s}{\Delta'}) \cdot x \cdot \frac{1}{s}$$

$\rightarrow$

Quantization scaler, Absmax within the group

$$\text{Err}(Q(w) \cdot x) = \Delta \cdot \text{Err}(\text{Round}(\frac{w}{\Delta})) \cdot x$$

$$\approx$$

$$\text{Err}(Q(w \cdot s) \cdot \frac{x}{s}) = \Delta' \cdot \text{Err}(\text{Round}(\frac{w \cdot s}{\Delta'})) \cdot x \cdot \frac{1}{s}$$

A scalar with an expectation of 0.25

- We assume weights are quantized to signed INT4 numbers in this analysis.
- As long as AWQ scales **s** is not too large (maintaining $\Delta \approx \Delta'$, since scaling up a single channel will usually not change the global absmax), quantization error is inversely proportional to **s**.

AWQ for low-bit weight-only quantization

Activation-aware optimal scaling searching

Quantize

fuse to previous op

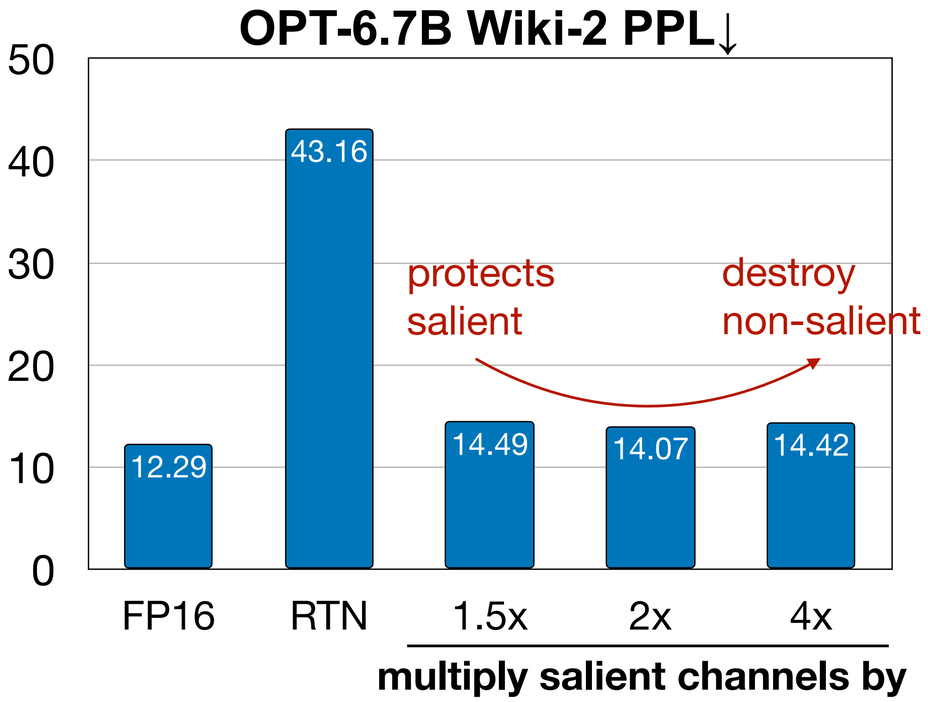
$$\mathbf{WX} \longrightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$

Activation diff, not weight diff

$$\mathscr{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{WX}\|$$

$$\mathbf{s} = \mathbf{s}_{\mathbf{X}}^{\alpha}, \alpha^* = \text{argmin}_{\alpha} \mathscr{L}(\mathbf{s}_{\mathbf{X}}^{\alpha})$$

avg. activation magnitude

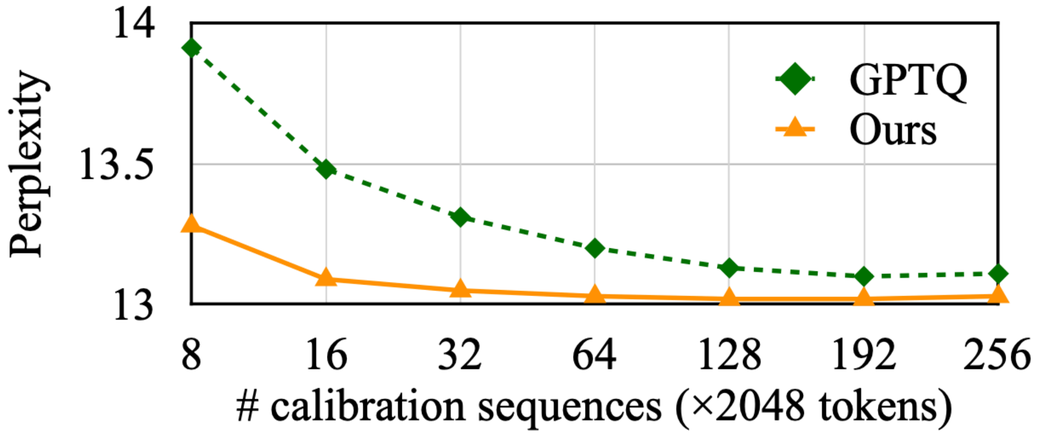


- Scale s is only determined by the average activation magnitude (**activation awareness**).
- The search objective is mean square root error for the **activation**, not the **weights** themselves (as in GPTQ).

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

Advantages of AWQ

- Accurate, simple and easy to implement
- High hardware efficiency
- Less dependency on calibration set compared to regression-based method
  - Better data efficiency and distribution robustness
  - Generalize to instruction-tuned model and multi-modal LMs (different distributions)



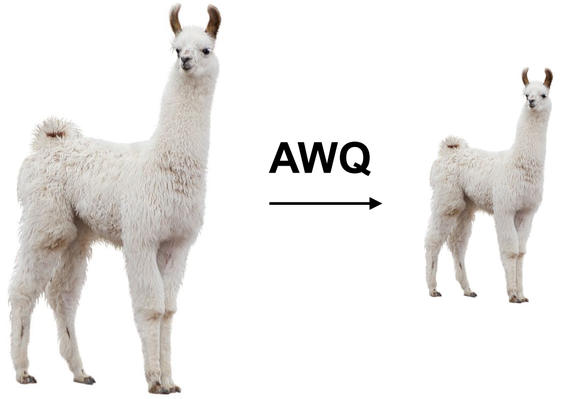
(a) Our method needs a smaller calibration set

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

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

AWQ results

Improving general LLM quantization

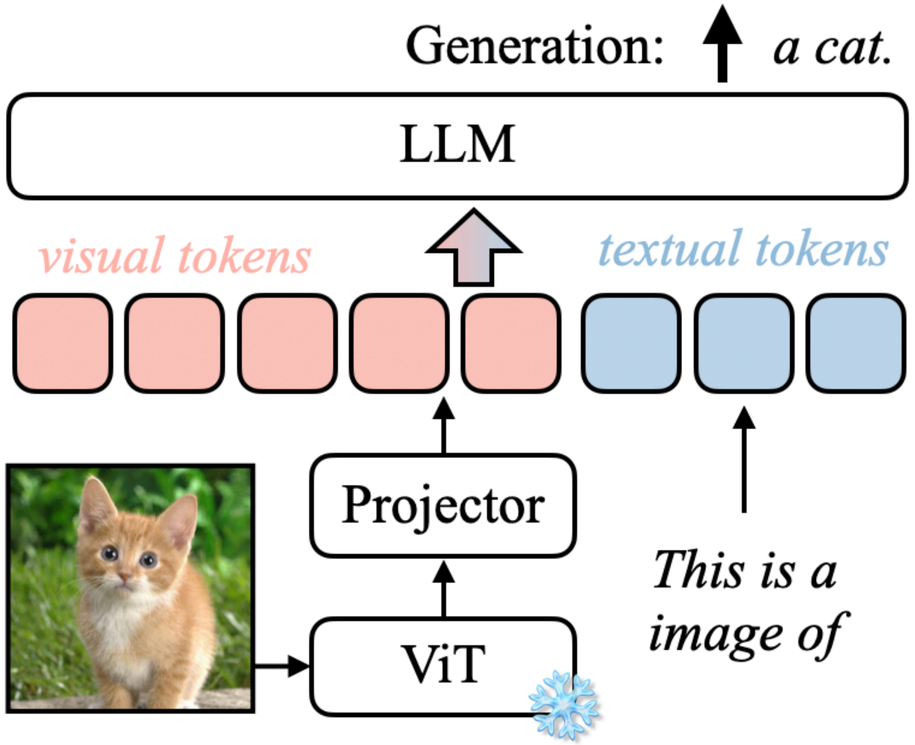


PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	**6.24**	**5.32**	**3.74**	**6.35**	**5.52**	**4.61**	**3.95**
INT4 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**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

AWQ for visual language model

- Goal: multi-modal LLM with excellent *visual-language* performance and *text-only* performance
- We study PaLM-E alike architecture due to its flexibility for multi-modal input/output
- Augment LLM with tokenized visual input



VILA: On Pre-training for Visual Language Models [Lin et al., CVPR 2024]

AWQ for visual language model

AWQ well preserves the accuracy of GPT-4o-like multi-modal LLM

Q: Elaborate on the visual and narrative elements of the video in detail.

A: The video shows a person's hands working on a white surface. They are folding a piece of fabric with a checkered pattern in shades of blue and white. The fabric is being folded into a smaller, more compact shape. The person's fingernails are painted red, and they are wearing a black and red garment. There are also a ruler and a pencil on the surface, suggesting that measurements and precision are involved in the process.

Demo: https://vila.hanlab.ai

VILA: On Pre-training for Visual Language Models [Lin et al., CVPR 2024]

AWQ results for visual language models

Quantization of multi-modal LMs (VILA)

Model	Prec.	VQAv2	GQA	VizWiz	SQA-I	VQA-T	POPE	MME	MMB	MMB-CN	SEED	llava-bench	MMVet	Average
VILA-7B	FP16	80.3	63.1	59.6	68.0	62.6	86.3	1489.4	69.8	61.0	61.7	75.2	35.1	65.7
VILA-7B-AWQ	INT4	80.1	63.0	57.8	68.0	61.9	85.3	1486.3	68.8	59.0	61.3	75.8	35.9	65.2
VILA-13B	FP16	80.5	63.6	63.1	70.5	64.0	86.3	1553.6	73.8	66.7	62.8	78.3	42.6	68.4
VILA-13B-AWQ	INT4	80.4	63.6	63.0	71.2	63.5	87.0	1552.9	73.6	66.3	62.2	77.6	42.0	68.2

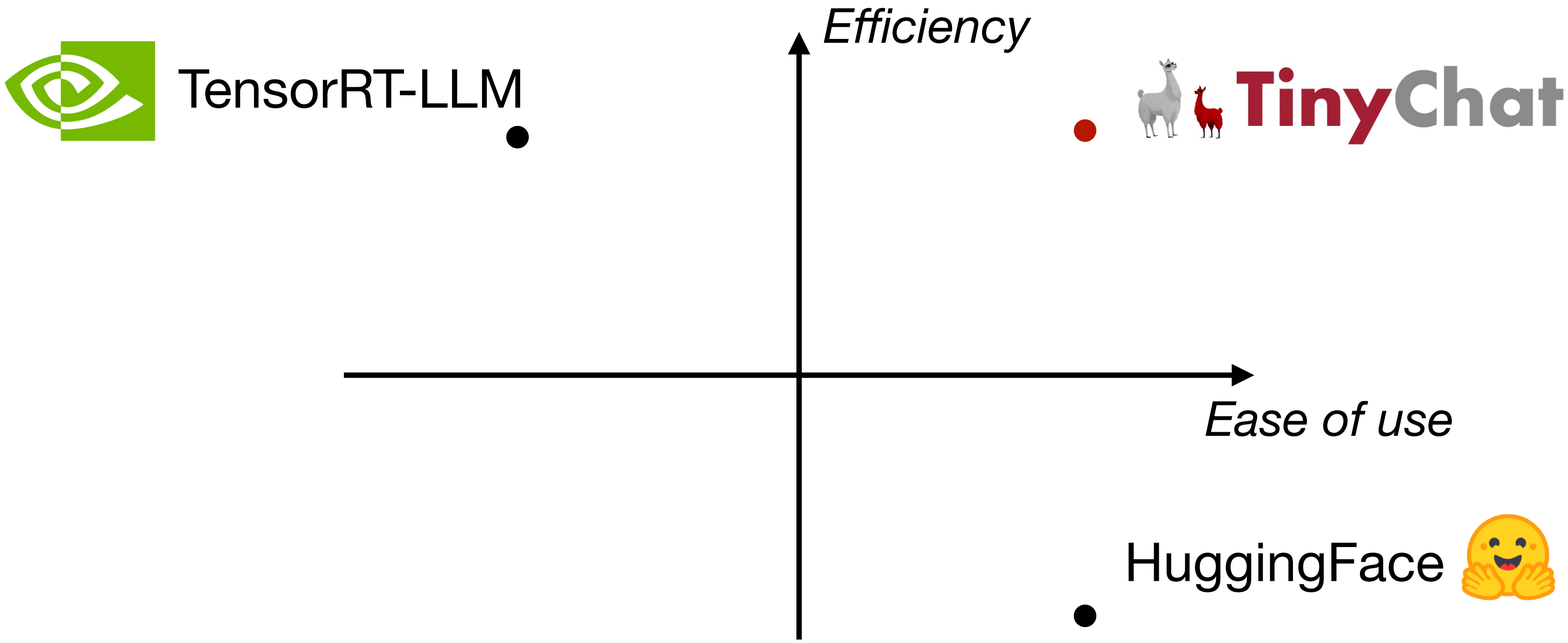
VILA: On Pre-training for Visual Language Models [Lin et al., CVPR 2024]

TinyChat : LLM Inference Engine on Edge

TinyChat: a lightweight LLM inference engine

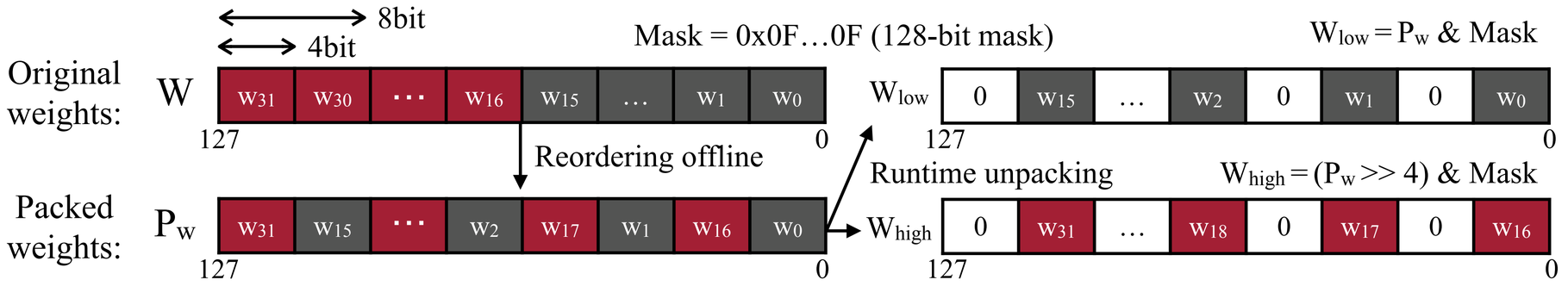
Pythonic, lightweight, efficient, multi-platform

- We need a framework to serve the quantized model to achieve low latency
  - HuggingFace: easy to use, but slow
  - TensorRT-LLM: high efficiency, but less flexibility
- **TinyChat**: efficient, lightweight, Python-native (composable with other stacks like vLLM), multi-platform (cloud, desktop/laptop, edge GPUs; desktop/laptop, mobile CPUs)

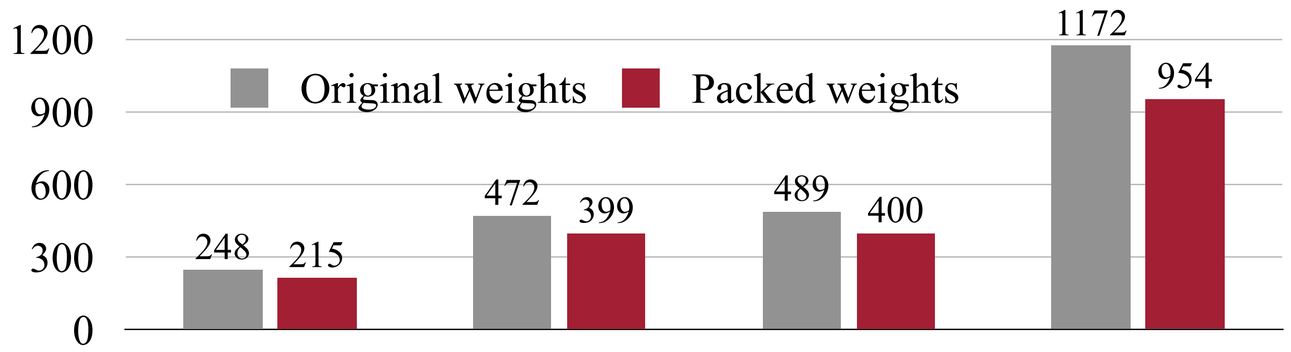


Hardware-aware packing

Reducing number of logical instructions to decode weights



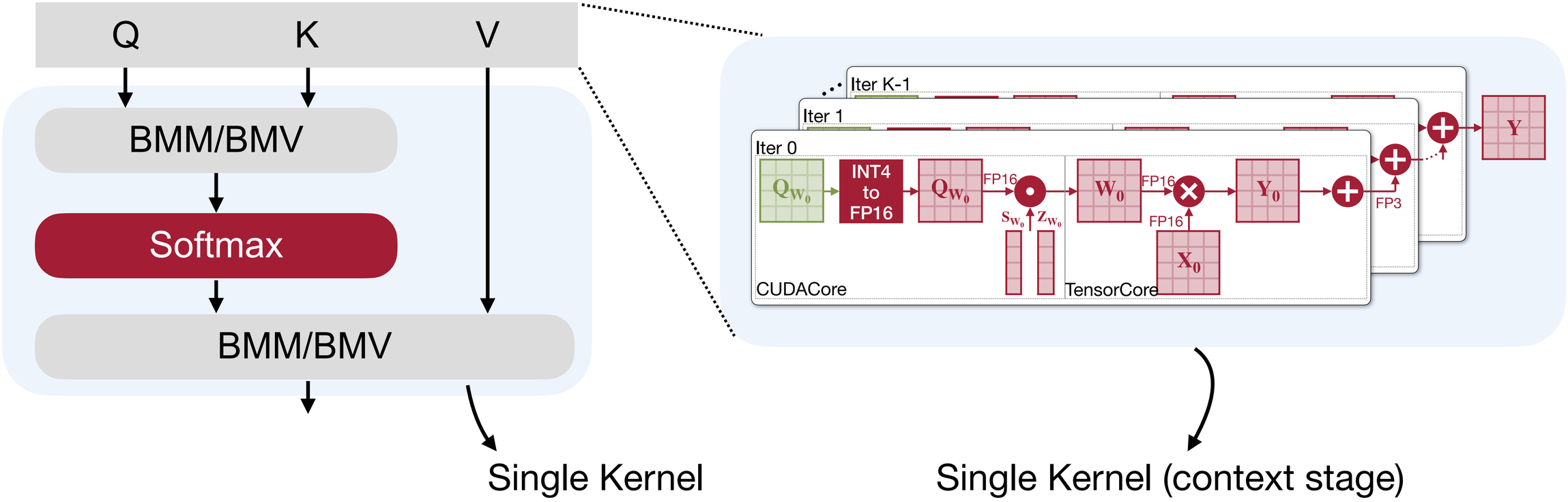
Decode 32 weights with only 3 logical operations



1.2x speedup on ARM NEON

Kernel fusion

Reduce intermediate DRAM access in memory bound ops



Single Kernel

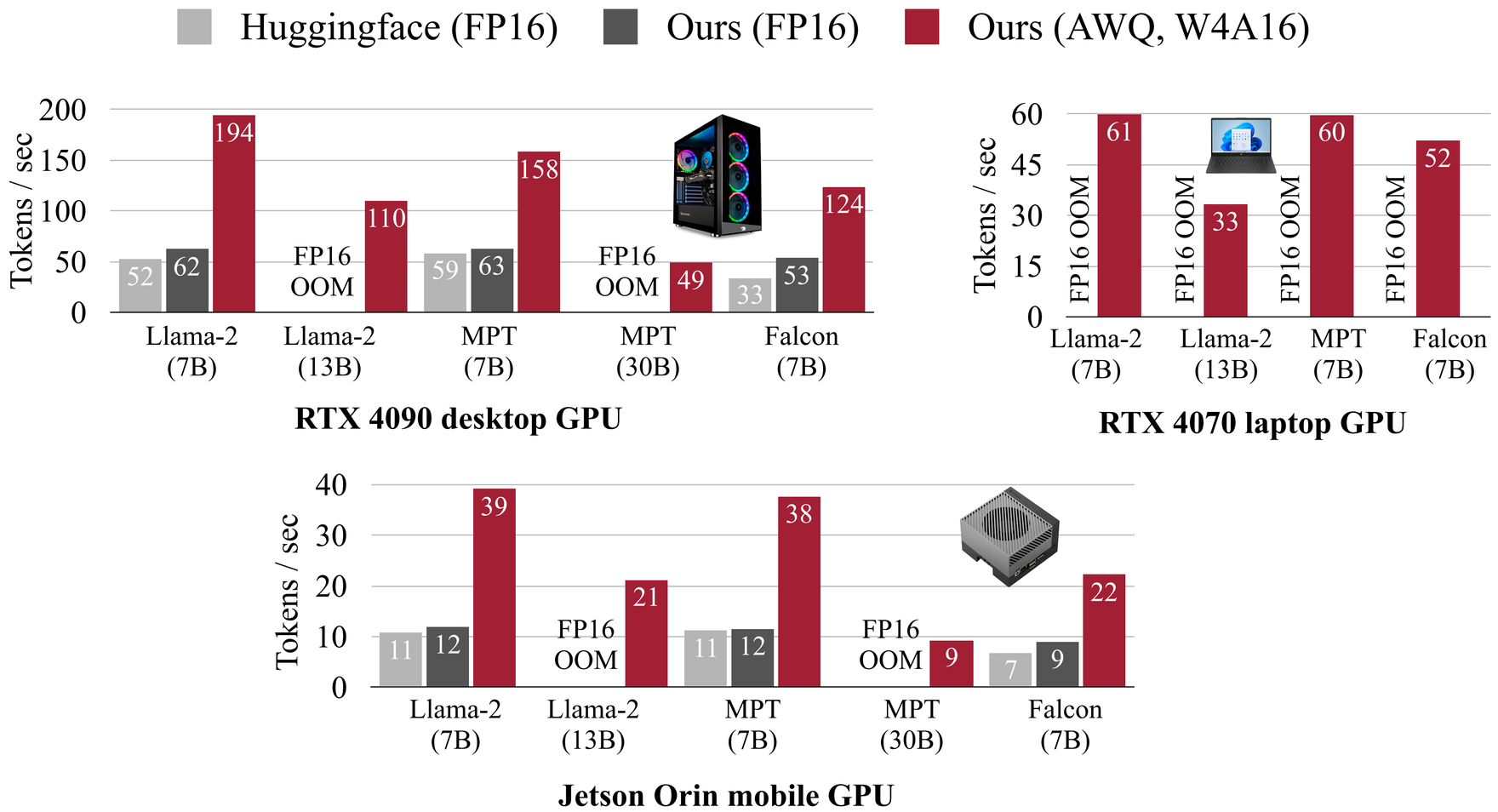
Fused kernel for attention

Single Kernel (context stage)

Fused kernel for GEMM

TinyChat significantly accelerates edge LLMs

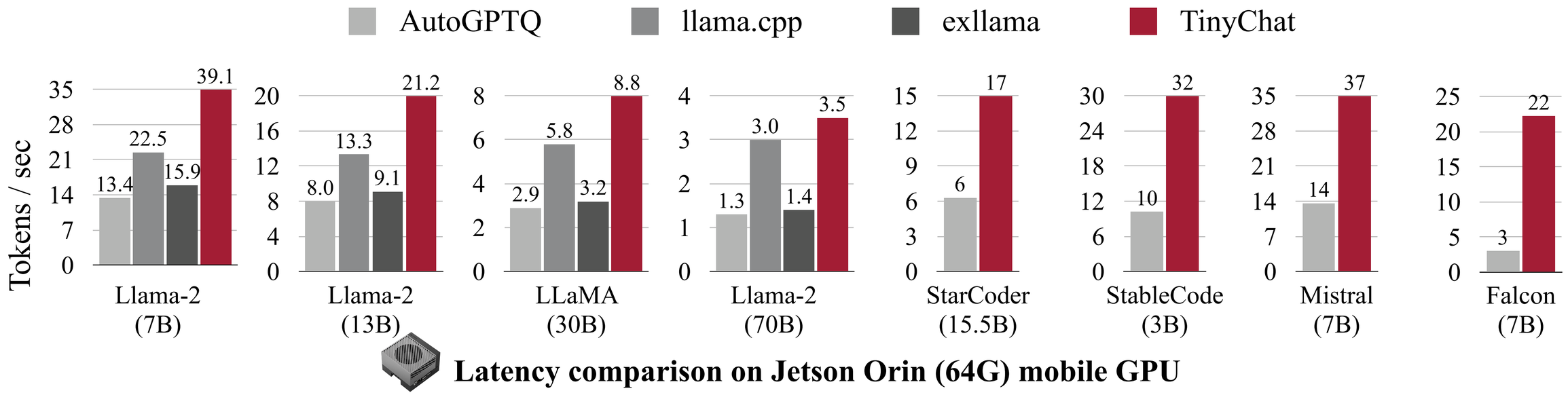
More than 3x faster than Huggingface FP16 LLM inference



AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

TinyChat outperforms other systems

At least 2.6x faster than AutoGPTQ and more flexible than llama.cpp, exllama



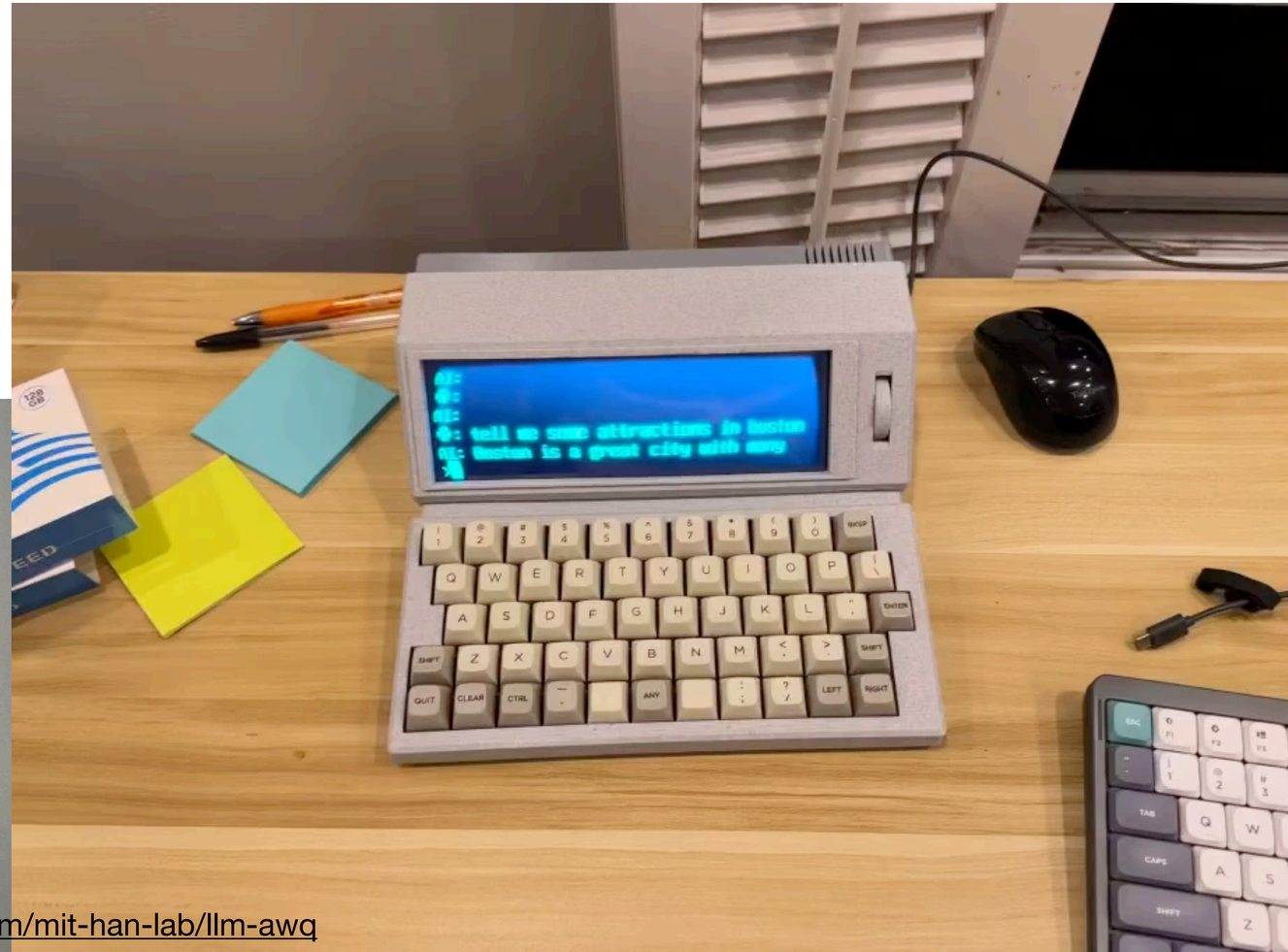
AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

TinyChat: a lightweight LLM inference engine

Demo on TinyChatComputer, powered by NVIDIA Jetson Orin Nano

- On a GPU board with just ~7G available memory, TinyChat enables efficient deployment of 7B large language models, thanks to AWQ quantization.

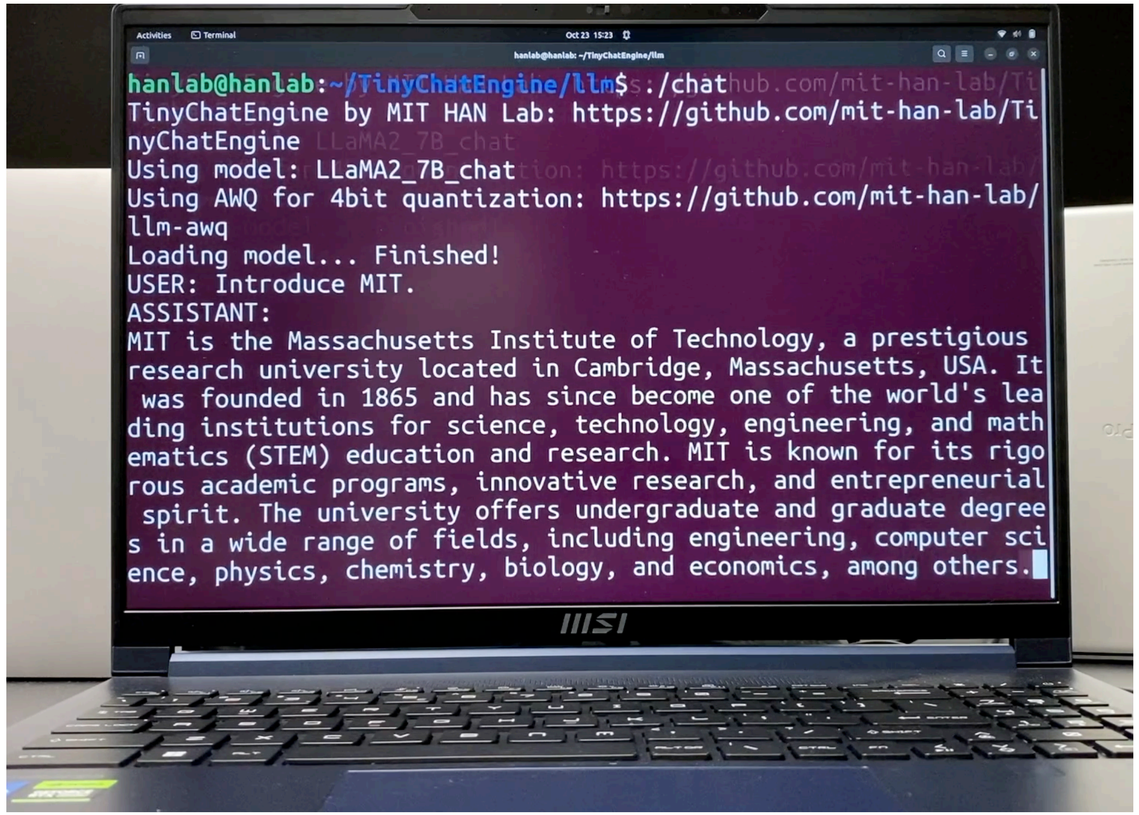
- Importance of edge LLM deployment: **data privacy and security**, **continued operation in disconnected environment**, **customization and personalization**.



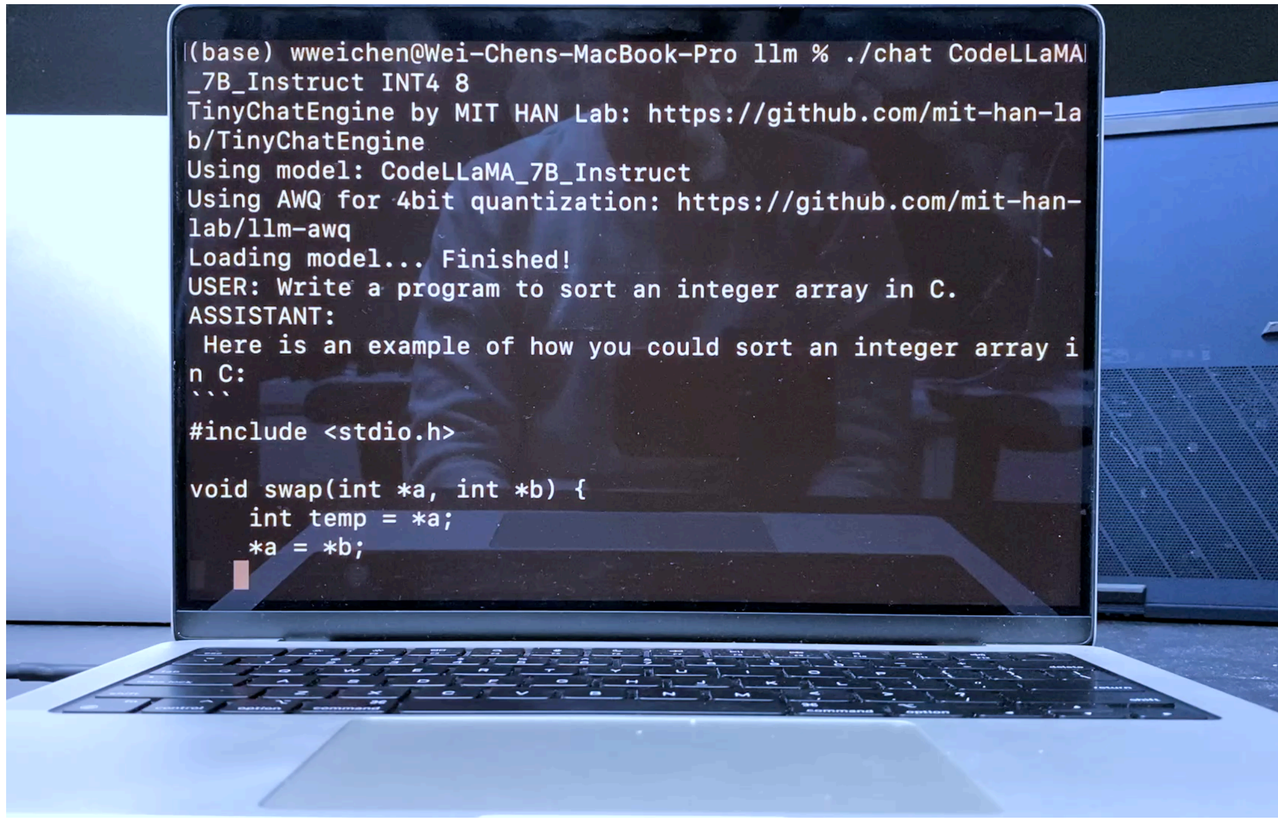
https://github.com/mit-han-lab/llm-awq

TinyChat: a lightweight LLM inference engine

TinyChat seamlessly supports personal laptops with Intel / ARM CPUs



MSI Laptop (RTX 4070 GPU)



Macbook (ARM CPU)

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

TinyChat seamlessly supports VLMs

Accelerating visual-language models across different GPU platforms

- TinyChat also seamlessly supports VILA, delivering ~3x speedup over FP16 on Orin and allows interactive VLM deployment on the edge (laptops and AIoT).

Model	Precision	A100 Tok/sec	4090 Tok / sec	Orin Tok / sec
Llama-3-VILA1.5-8B	FP16	74.9	57.4	10.2
Llama-3-VILA1.5-8B-AWQ	INT4	**168.9**	**150.2**	**28.7**
VILA1.5-13B	FP16	50.0	OOM	6.1
VILA1.5-13B-AWQ	INT4	**115.9**	**105.7**	**20.6**

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

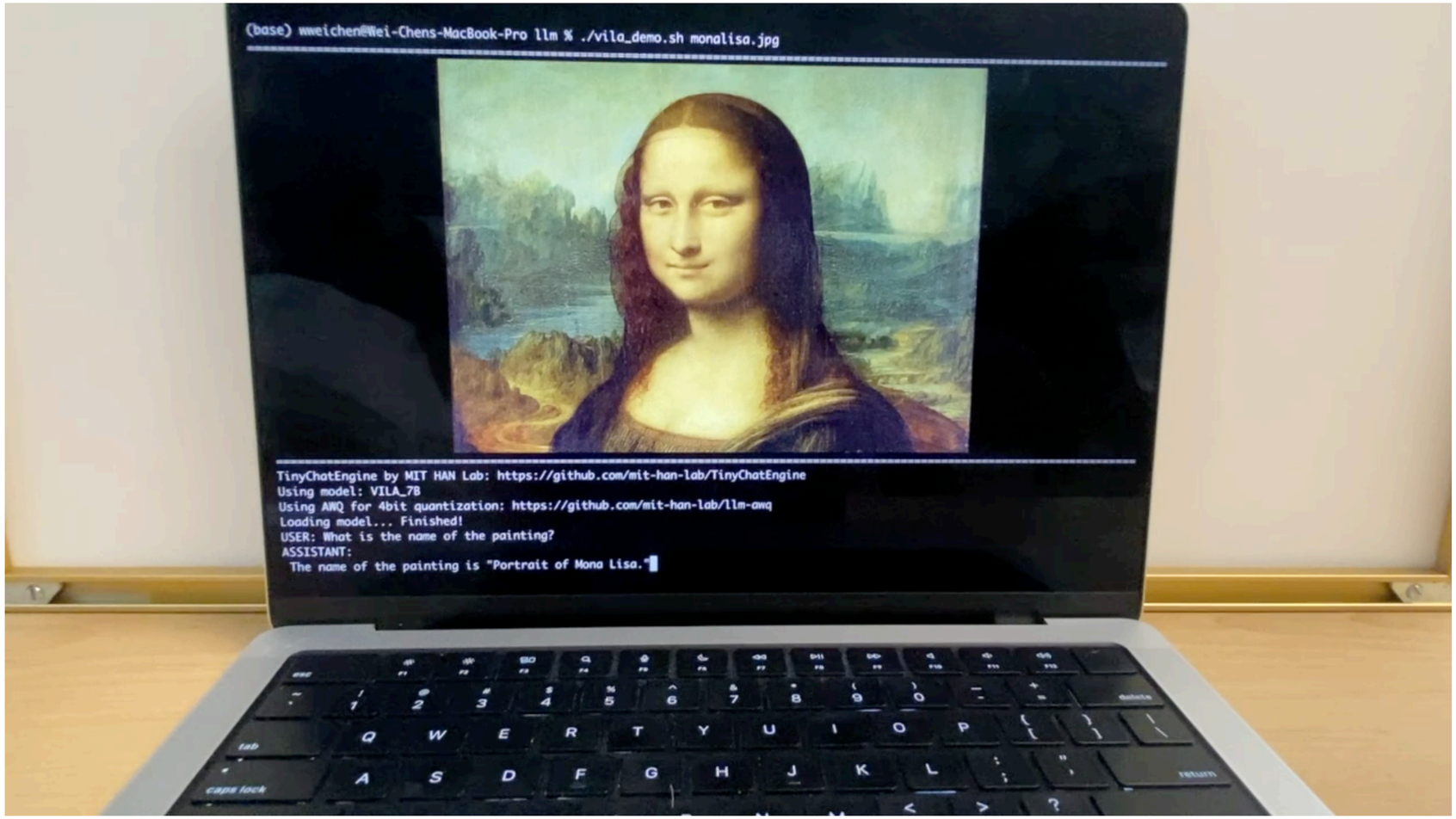
TinyChat for visual language model

Efficient image reasoning on Jetson Orin: TinyChat w/ VILA model family

VILA: On Pre-training for Visual Language Models [Lin et al., CVPR 2024]

TinyChat : a lightweight LLM inference engine

Run visual language models on personal laptops



VILA-7B + AWQ: Running on MacBook Arm CPU

VILA: On Pre-training for Visual Language Models [Lin et al., CVPR 2024]

Community impact of AWQ

AWQ quantized models are downloaded 1 million times on HuggingFace

NVIDIA TensorRT-LLM https://github.com/NVIDIA/TensorRT-LLM#key-features

 Transformer Quantization API https://huggingface.co/docs/transformers/main_classes/quantization

IBM Granite IBM's internal code model, Granite, utilizes AWQ for quantization.

 lmdeploy https://github.com/InternLM/lmdeploy/blob/main/lmdeploy/lite/quantization/awq.py

Berkeley UNIVERSITY OF CALIFORNIA vLLM https://github.com/vllm-project/vllm/blob/main/vllm/model_executor/layers/quantization/awq.py

 Vertex AI https://console.cloud.google.com/vertex-ai/publishers/google/model-garden

Google Cloud

 lm-sys/FastChat https://github.com/lm-sys/FastChat/blob/main/docs/awq.md

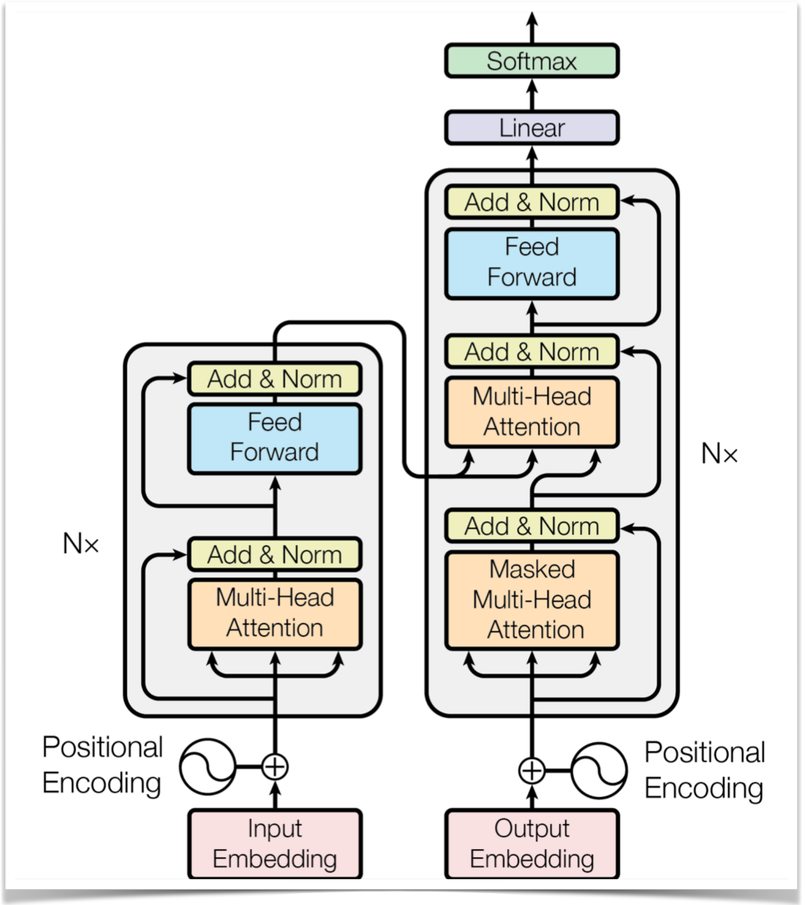
intel® https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration [Lin et al., MLSys 2024]

Lecture Plan

Today, we will cover:

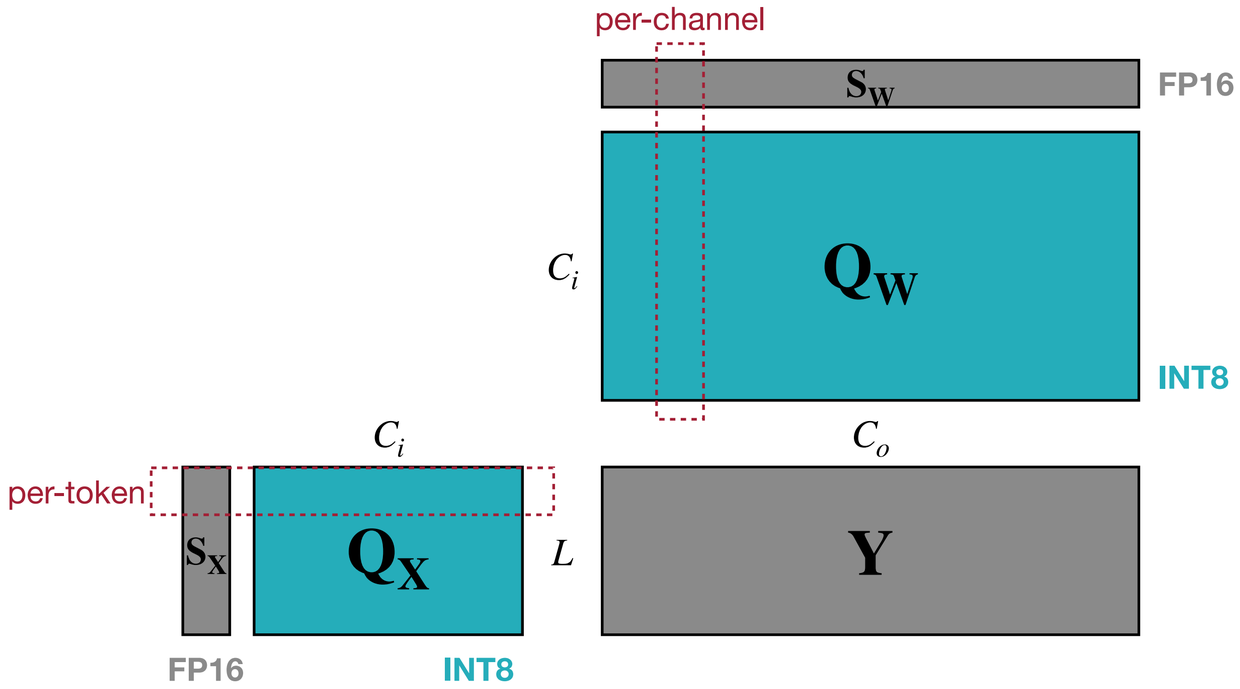
1. Linear Quantization Basics
2. Weight-Activation Quantization: SmoothQuant
3. Weight-Only Quantization: AWQ and TinyChat
4. **Pushing Further: QServe (W4A8KV4)**

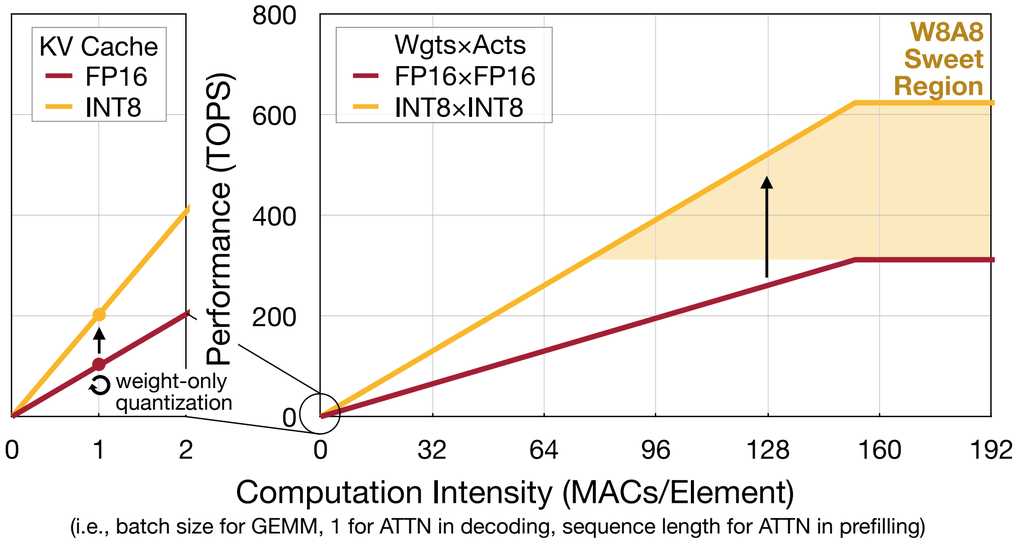


Inconsistency b/w Cloud and Edge Inference

W8-A8-KV8 for cloud serving while W4-A16-KV16 for edge inference.

- For cloud serving, the most popular quantization setting is W8-A8-QKV8
  - Weights: 8-bit Per-Channel Quantization
  - Activations: 8-bit Per-Token Quantization
  - Q-K-V: 8-bit Per-Tensor Quantization

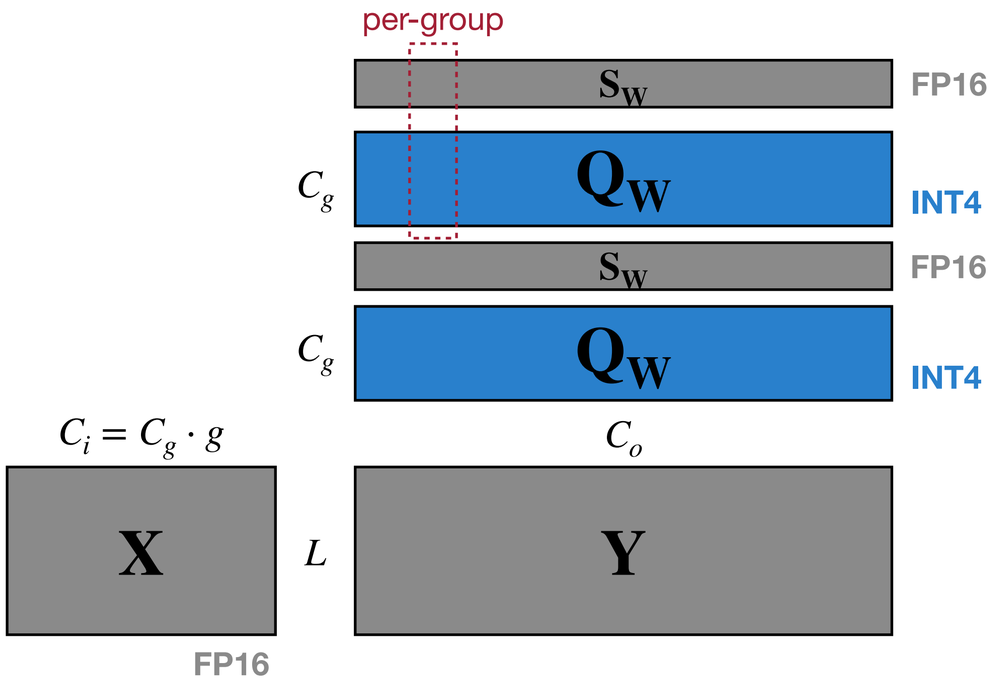


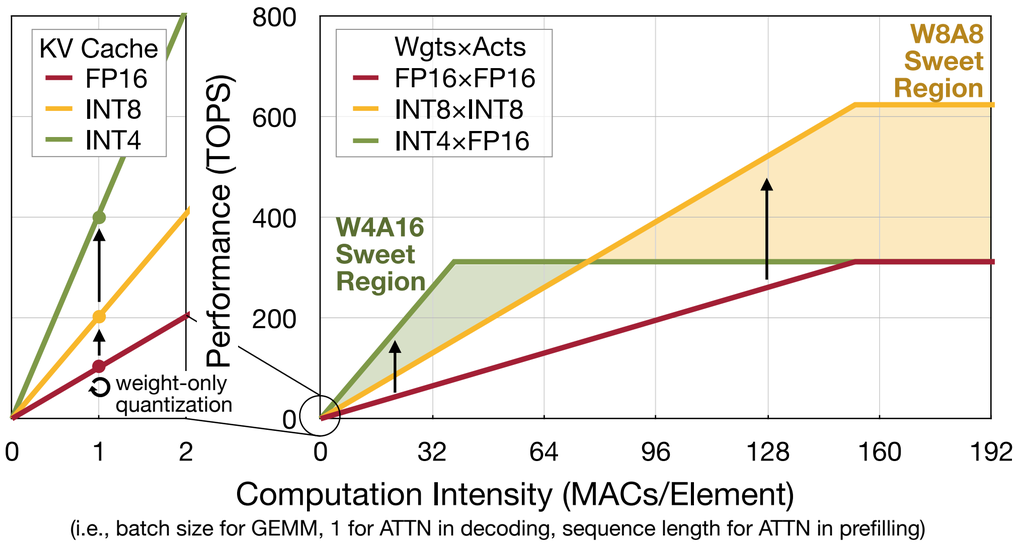


Inconsistency b/w Cloud and Edge Inference

W8-A8-KV8 for cloud serving while W4-A16-KV16 for edge inference.

- For edge inference, the most popular quantization setting is W4-A16-QKV16
  - Weights: 4-bit Per-Group Quantization ($C_g = 128$)
  - Activations: keeps FP16
  - Q-K-V: keeps FP16

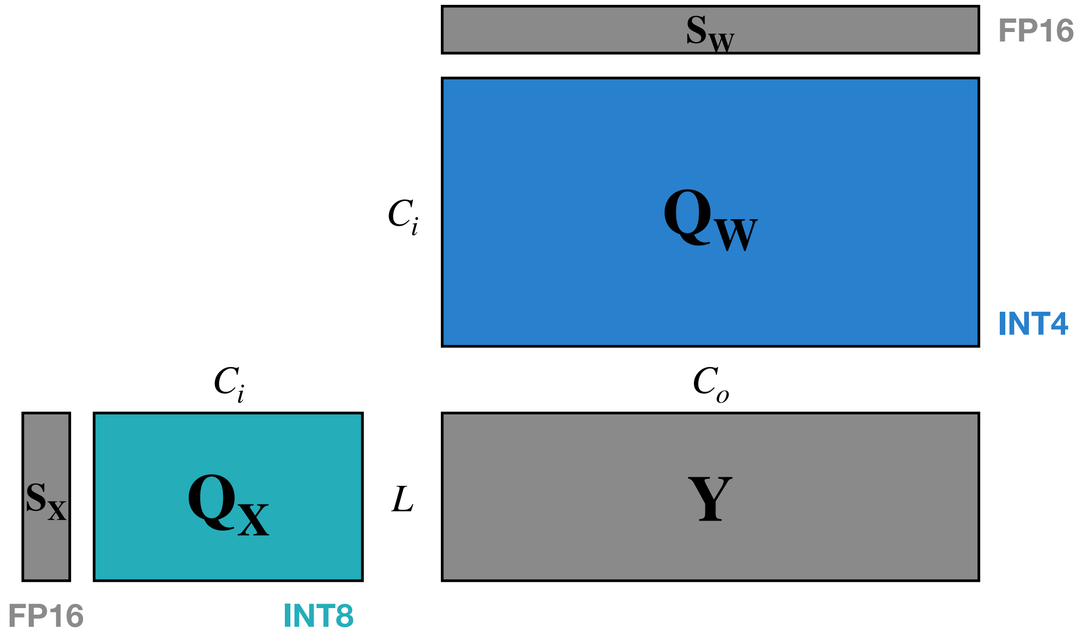


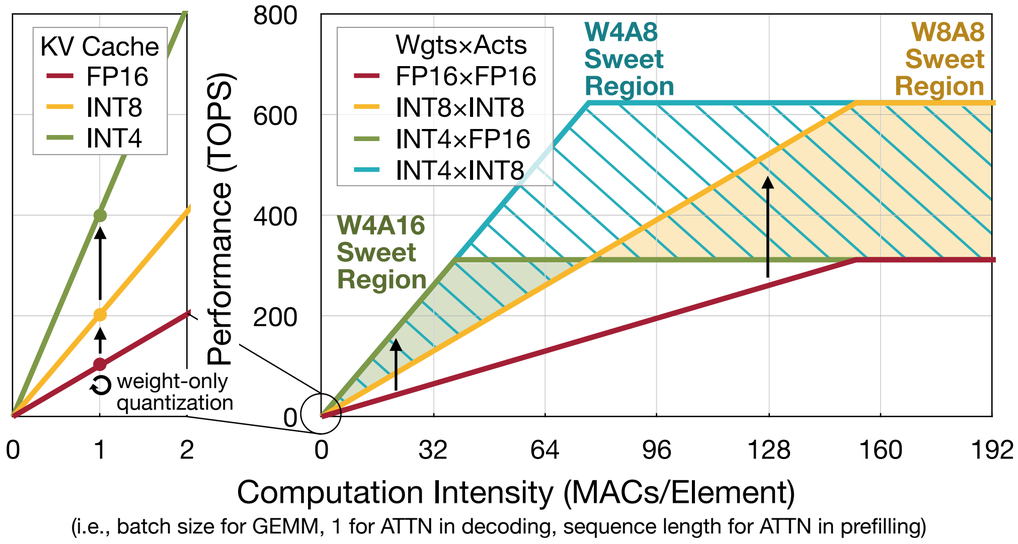


Inconsistency b/w Cloud and Edge Inference

W8-A8-KV8 for cloud serving while W4-A16-KV16 for edge inference.

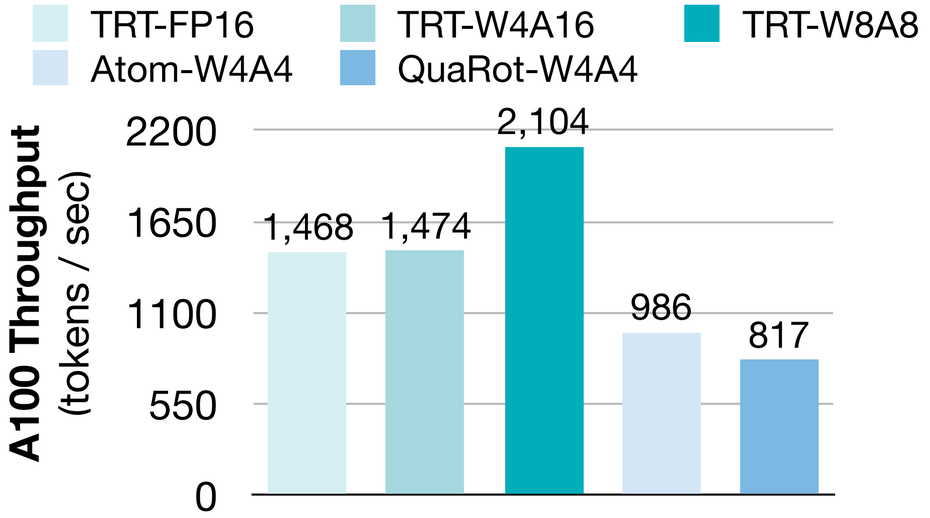
- Can we combine the advantage of both worlds?
  - Using 4-bit weights to save memory bandwidth
  - Using 8-bit activations to improve peak performance





Current status of LLM serving systems

W8A8 provides the highest throughput for cloud serving.



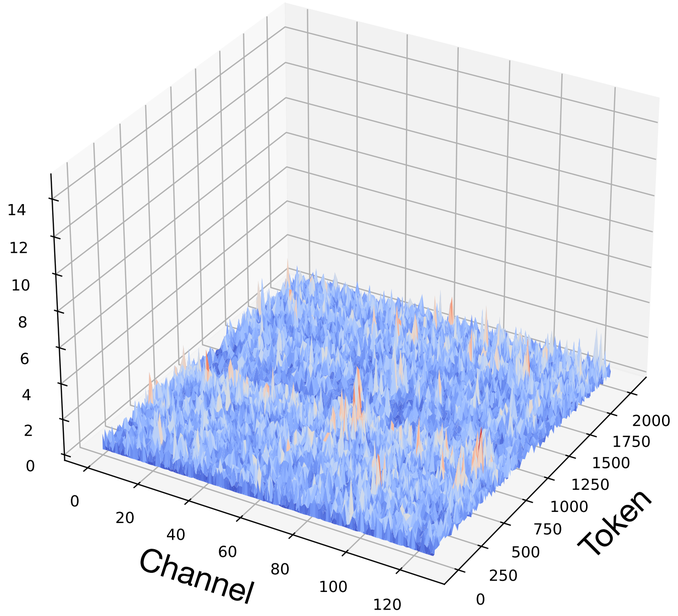
(b) Llama-2-7B system throughput on A100

Despite the existence of aggressive W4A4 quantization algorithms, they:

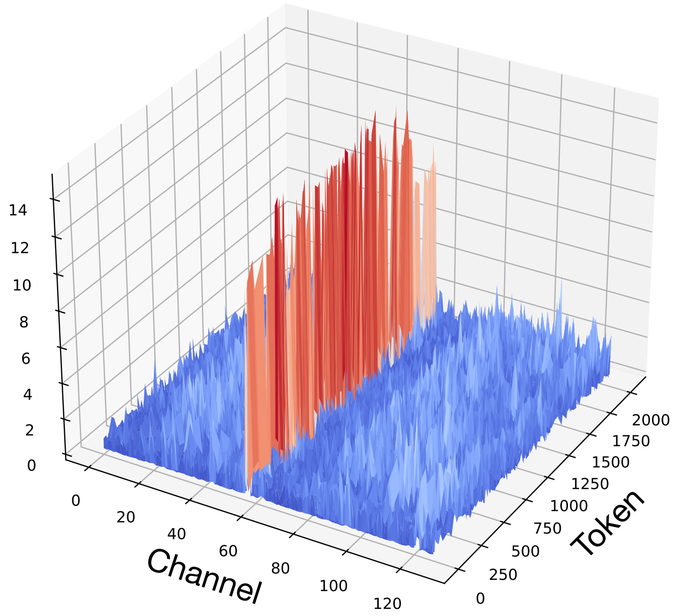
1. Bring about **significant accuracy loss**;
2. **Cannot run efficiently** on current GPUs.

SmoothAttention

Migrate quantization difficulty from K cache to Q matrix



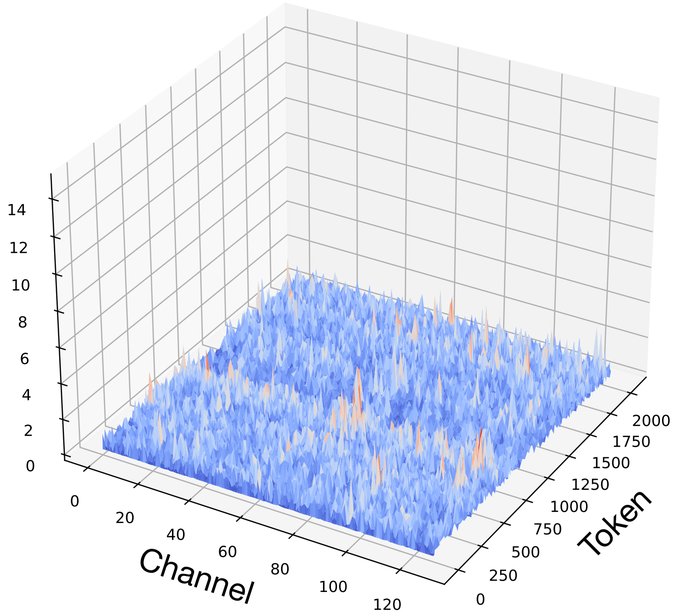
Layer 24 Values



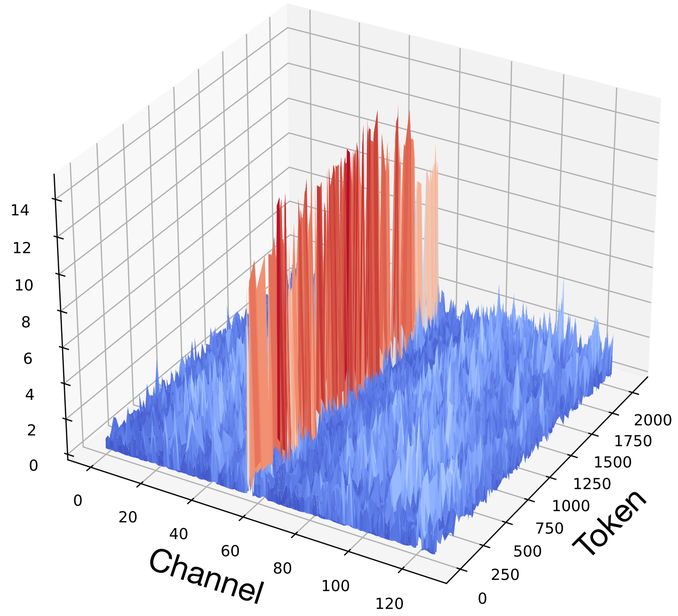
Layer 24 post-RoPE Keys
(Original)

SmoothAttention

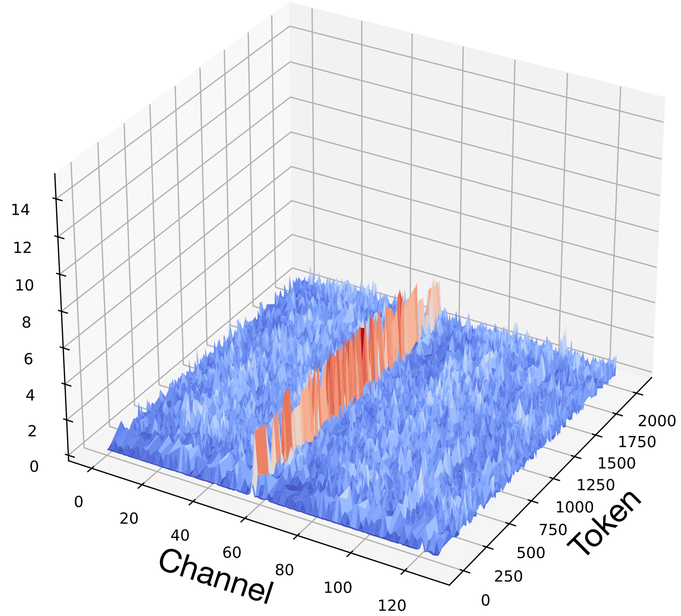
Migrate quantization difficulty from K cache to Q matrix



Layer 24 Values



Layer 24 post-RoPE Keys
(Original)



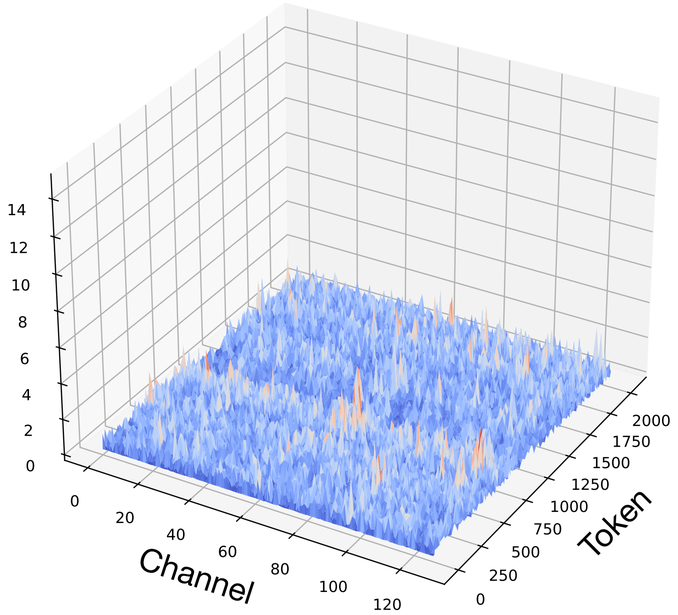
Layer 24 post-RoPE Keys
(SmoothAttention)

$$\mathbf{Z} = \left(\mathbf{Q}\mathbf{\Lambda}\right) \cdot \left(\mathbf{K}\mathbf{\Lambda}^{-1}\right)^{T}, \quad \mathbf{\Lambda} = \mathrm{diag}\left(\lambda\right)$$

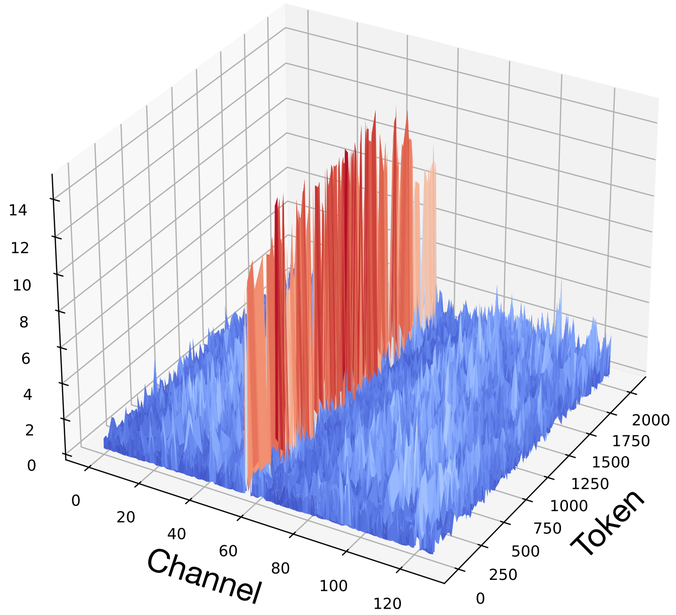
$$\lambda_i = \max\left(\left|\mathbf{K}_i\right|\right)^{\alpha}$$

SmoothAttention

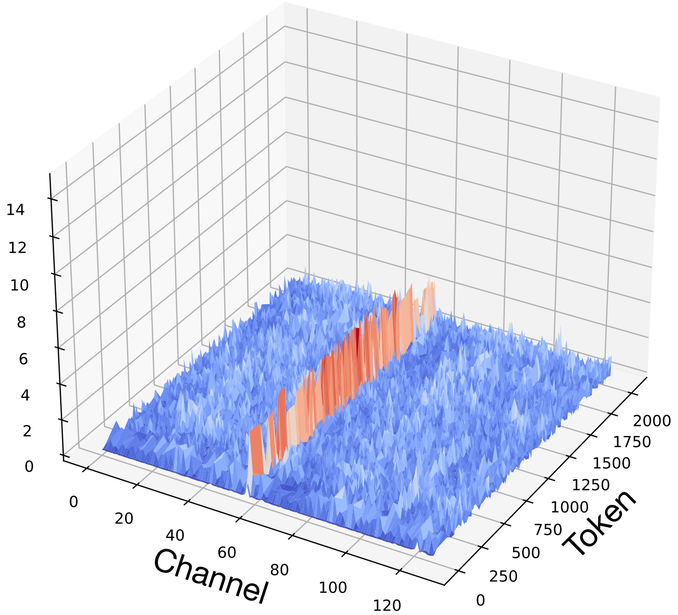
Migrate quantization difficulty from K cache to Q matrix



Layer 24 Values



Layer 24 post-RoPE Keys
(Original)

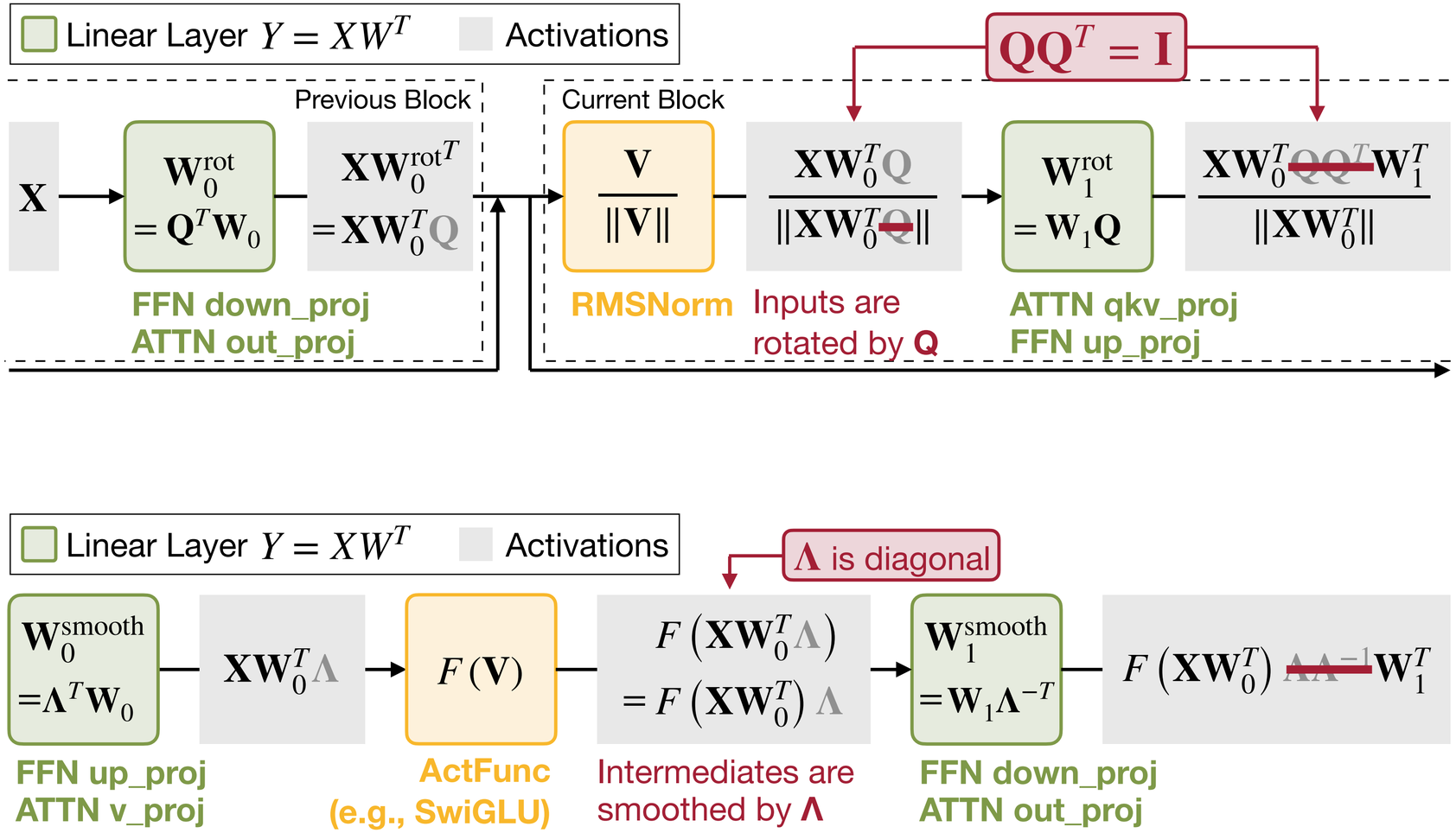


Layer 24 post-RoPE Keys
(SmoothAttention)

$$\mathbf{Z} = \left(\mathbf{Q}\boldsymbol{\Lambda}\right) \cdot \left(\mathbf{K}\boldsymbol{\Lambda}^{-1}\right)^{T}, \quad \boldsymbol{\Lambda} = \operatorname{diag}\left(\lambda\right)$$

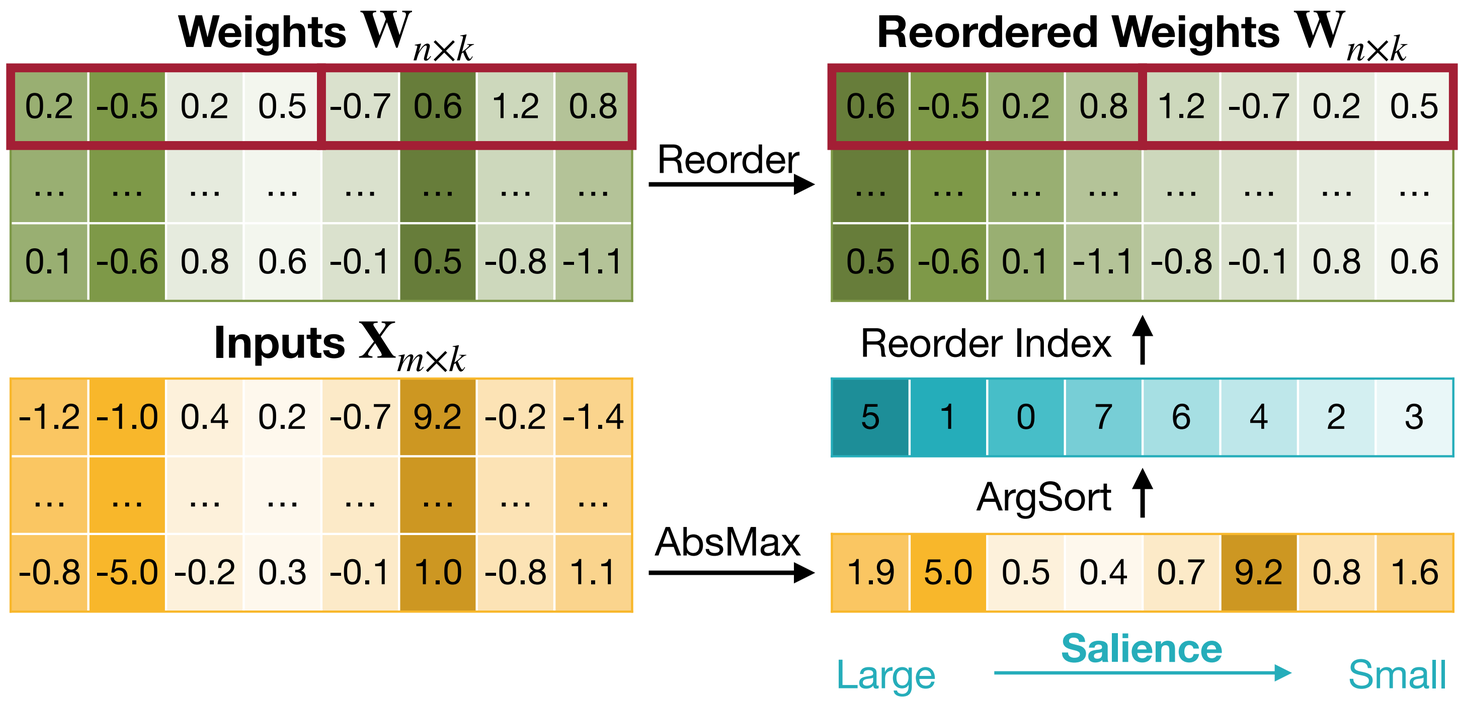
$$\lambda_i = \lambda_{i+\frac{D}{2}} = \max\left(\max\left(|\mathbf{K}_i|\right), \max\left(|\mathbf{K}_{i+\frac{D}{2}}|\right)\right)^{\alpha}$$

Rotation and Smoothing to suppress outliers



Activation-Aware Channel Reordering

- Maintaining the salient weights in FP16 can significantly improve model performance.
- These salient weights can be identified by the activation distribution.
- Instead of introducing mixed-precision quantization used by Atom, we propose activation-aware channel reordering.



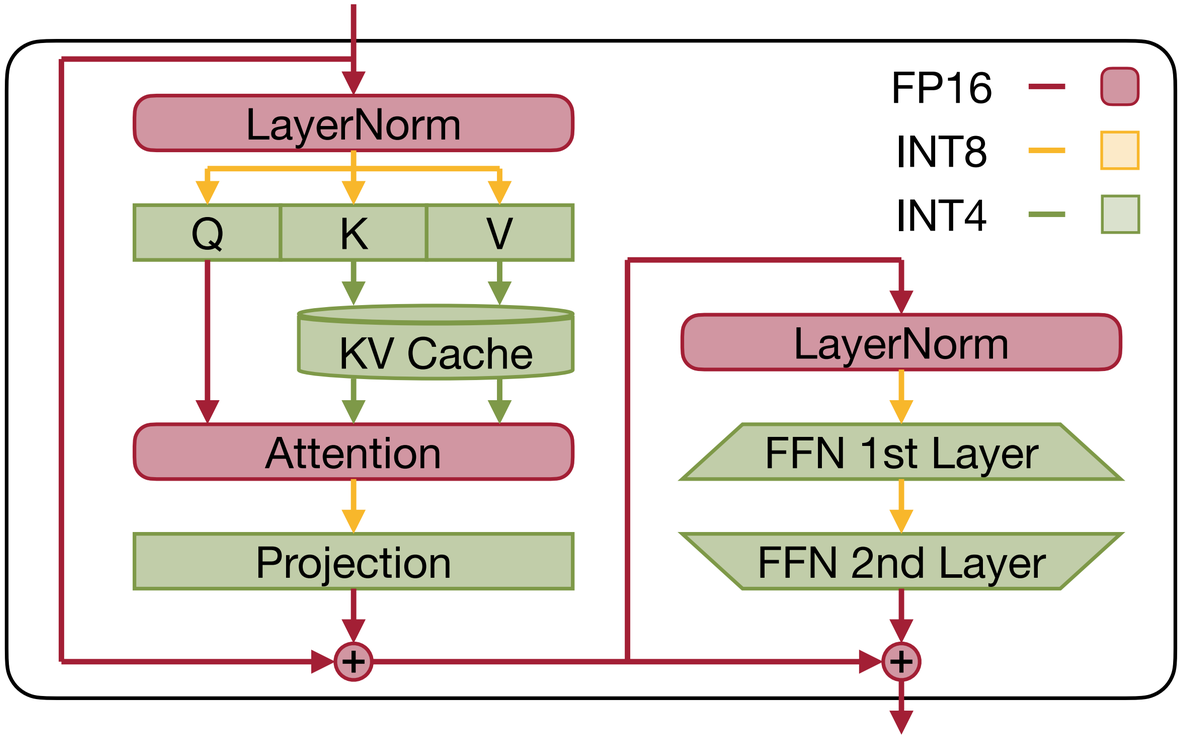
QoQ outperforms the state-of-the-art.

WikiText2 perplexity with 2048 sequence length. The lower is the better.

WikiText2 Perplexity ↓		Llama-2			Llama			Mistral	Mixtral	Yi
Precision	Algorithm	7B	13B	70B	7B	13B	30B	7B	8x7B	34B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	5.25	3.84	4.60
W8A8	SmoothQuant	5.54	4.95	3.36	5.73	5.13	4.23	5.29	3.89	4.69
W4A16 g128	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	5.39	4.08	4.68
	AWQ	5.60	4.97	3.41	5.78	5.19	4.21	5.37	4.02	4.67
W4A4	QuaRot	6.10	5.40	3.79	6.26	5.55	4.60	5.71	NaN	NaN
		6.19	5.45	3.83	6.34	5.58	4.64	5.77	NaN	NaN
W4A4 g128	QuaRot	5.93	5.26	3.61	6.06	5.40	4.44	5.54	NaN	NaN
		6.00	5.31	3.64	6.13	5.43	4.48	5.58	NaN	NaN
	Atom	6.03	5.27	3.69	6.16	5.46	4.55	5.66	4.42	4.92
		6.12	5.31	3.73	6.25	5.52	4.61	5.76	4.48	4.97
W4A8KV4	RTN	6.51	5.40	3.90	6.51	5.71	4.91	6.18	5.02	6.52
	AWQ	6.27	5.25	3.68	6.33	5.59	4.61	5.92	4.58	5.26
	Quarot	5.73	5.07	3.46	5.93	5.29	4.32	5.41	NaN	NaN
	Atom	5.91	5.16	3.60	6.03	5.41	4.49	5.55	NaN	4.84
	QServe	5.75	5.12	3.52	5.94	5.28	4.34	5.46	4.18	4.74
W4A8KV4 g128	RTN	5.99	5.19	3.70	6.23	5.46	4.56	5.59	4.39	5.49
	AWQ	5.83	5.12	3.51	5.93	5.36	4.39	5.50	4.23	4.78
	Quarot	5.71	**5.06**	**3.45**	5.91	5.26	4.30	**5.39**	NaN	NaN
	Atom	5.80	5.10	3.53	5.95	5.36	4.41	5.47	4.22	4.75
	QServe	**5.70**	**5.06**	3.47	**5.90**	**5.24**	**4.28**	5.41	**4.14**	**4.74**

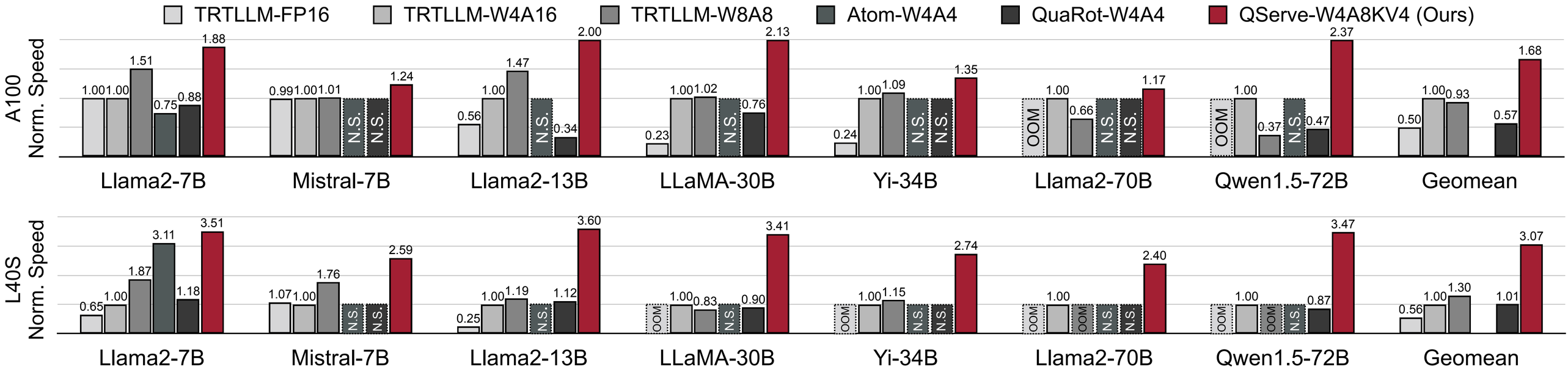
* Grayed results use Wikitext2 as calibaration dataset.

C


FP16
INT8
INT4
LayerNorm
Q
K
V
KV Cache
Attention
Projection
LayerNorm
FFN 1st Layer
FFN 2nd Layer

End-to-end Throughput Evaluation Results

Up to 2.4x-3.5x faster than TensorRT-LLM on A100, L40S



End-to-end Throughput Evaluation Results

Up to 2.4x-3.5x faster than TensorRT-LLM on A100, L40S

Device	System	Llama-2 7B	Mistral 7B	LLama-2 13B	LLaMA 30B	Yi 34B	Llama-2 70B	Qwen1.5 72B
A100	TRTLLM-Best	2334	2427	1277	361	649	358	143
	QServe (Ours)	2908	2970	1741	749	797	419	340
	Speedup	1.25×	1.22×	1.36×	2.07×	1.23×	1.17×	2.38×
L40S	TRTLLM-Best	1271	2569	440	148	364	119	17
	QServe (Ours)	2394	3774	1327	504	869	286	59
	Speedup	1.88×	1.47×	3.02×	3.41×	2.39×	2.40×	3.47×

Measured throughput (tokens/s) on A100 / L40S

Summary of Today's Lecture

1. Linear Quantization Basics
2. Weight-Activation Quantization: SmoothQuant
3. Weight-Only Quantization: AWQ and TinyChat
4. Pushing Further: QServe (W4A8KV4)