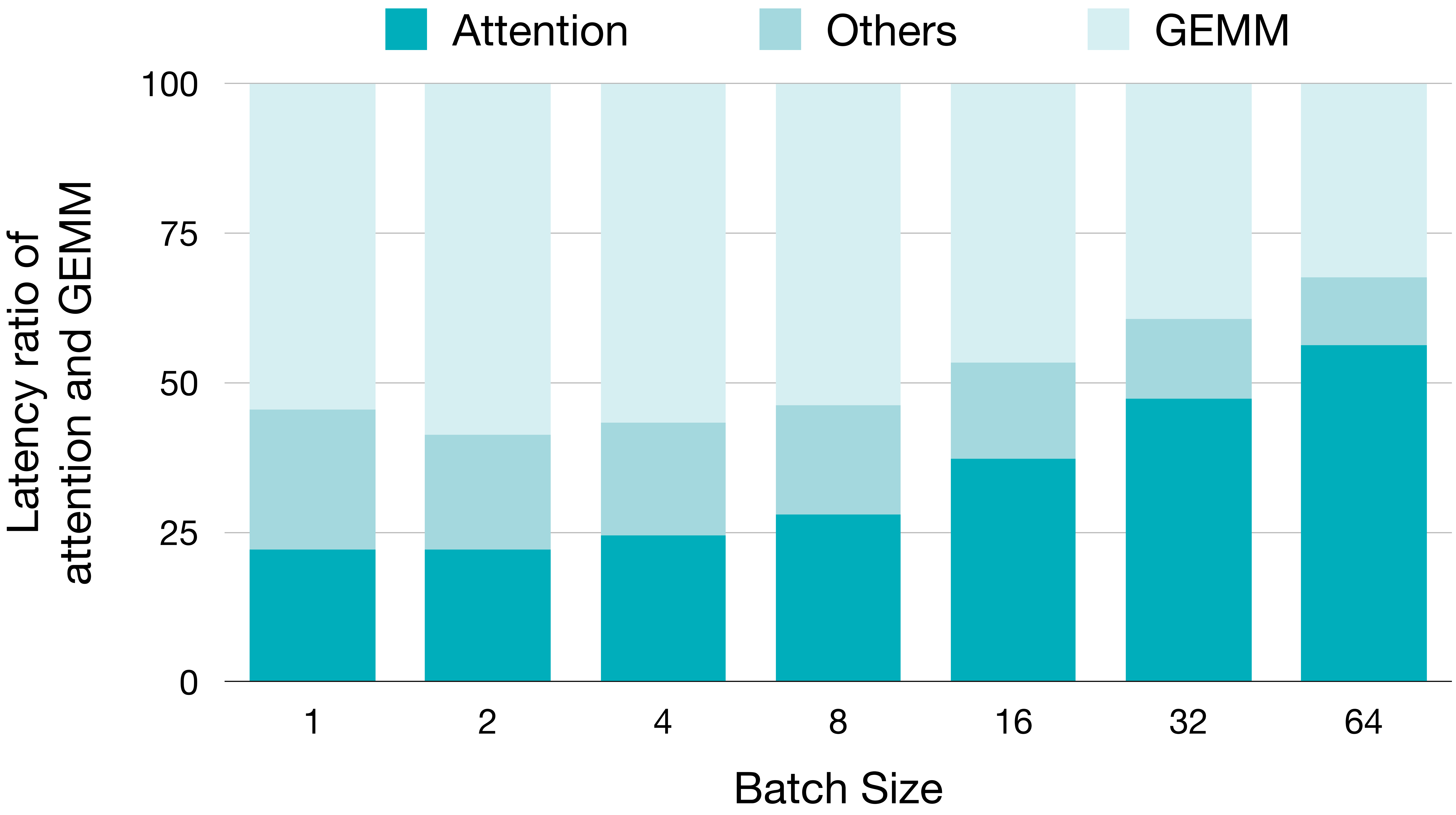
QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving

Yujun Lin*, Haotian Tang*, **Shang Yang***, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, Song Han
MIT, NVIDIA, MIT-IBM Watson AI Lab, UMass Amherst.
Accepted by MLSys 2025.

* indicates the equal contribution to this work.

Current status of LLM serving systems

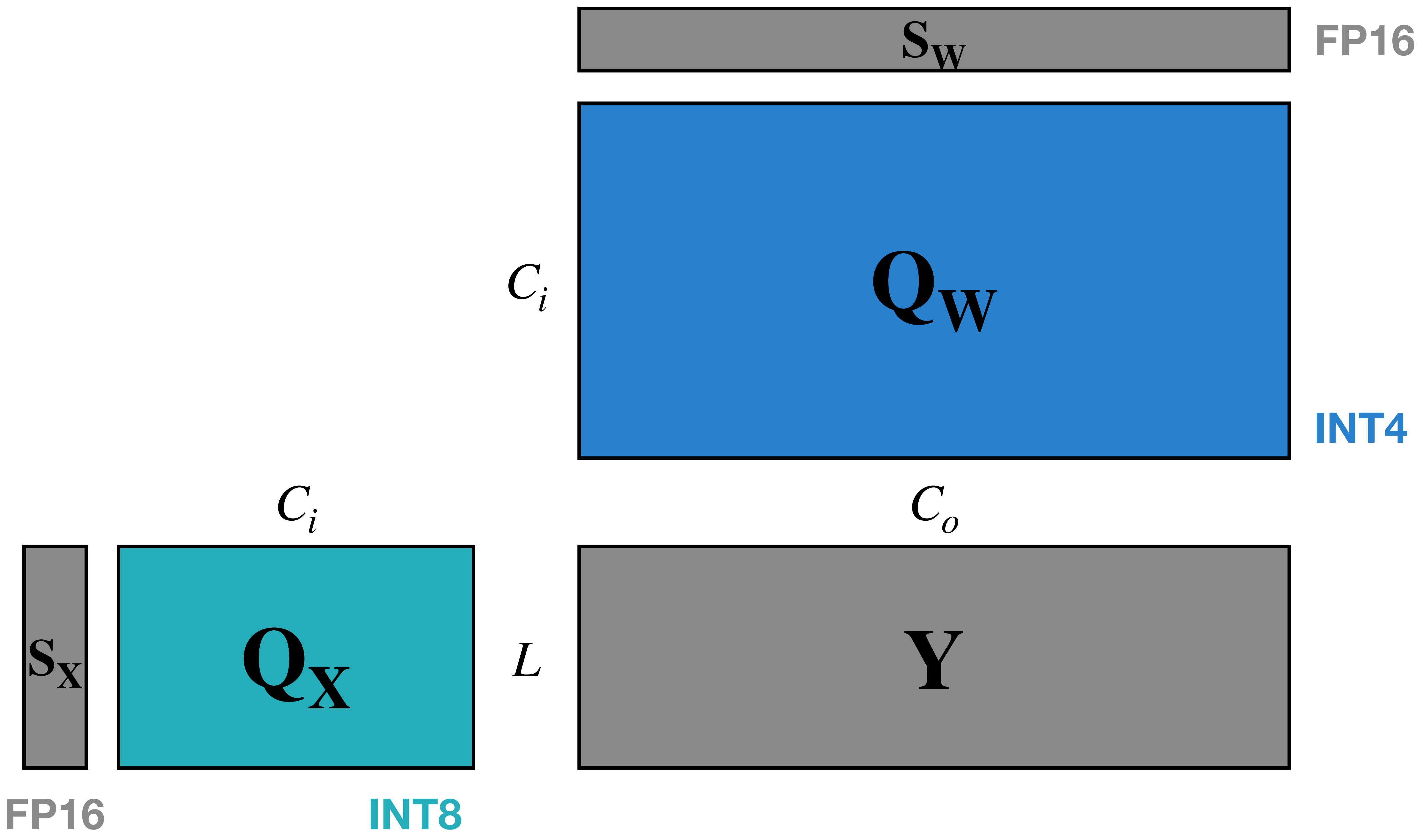
GEMM and Attention together dominate the LLM inference latency.

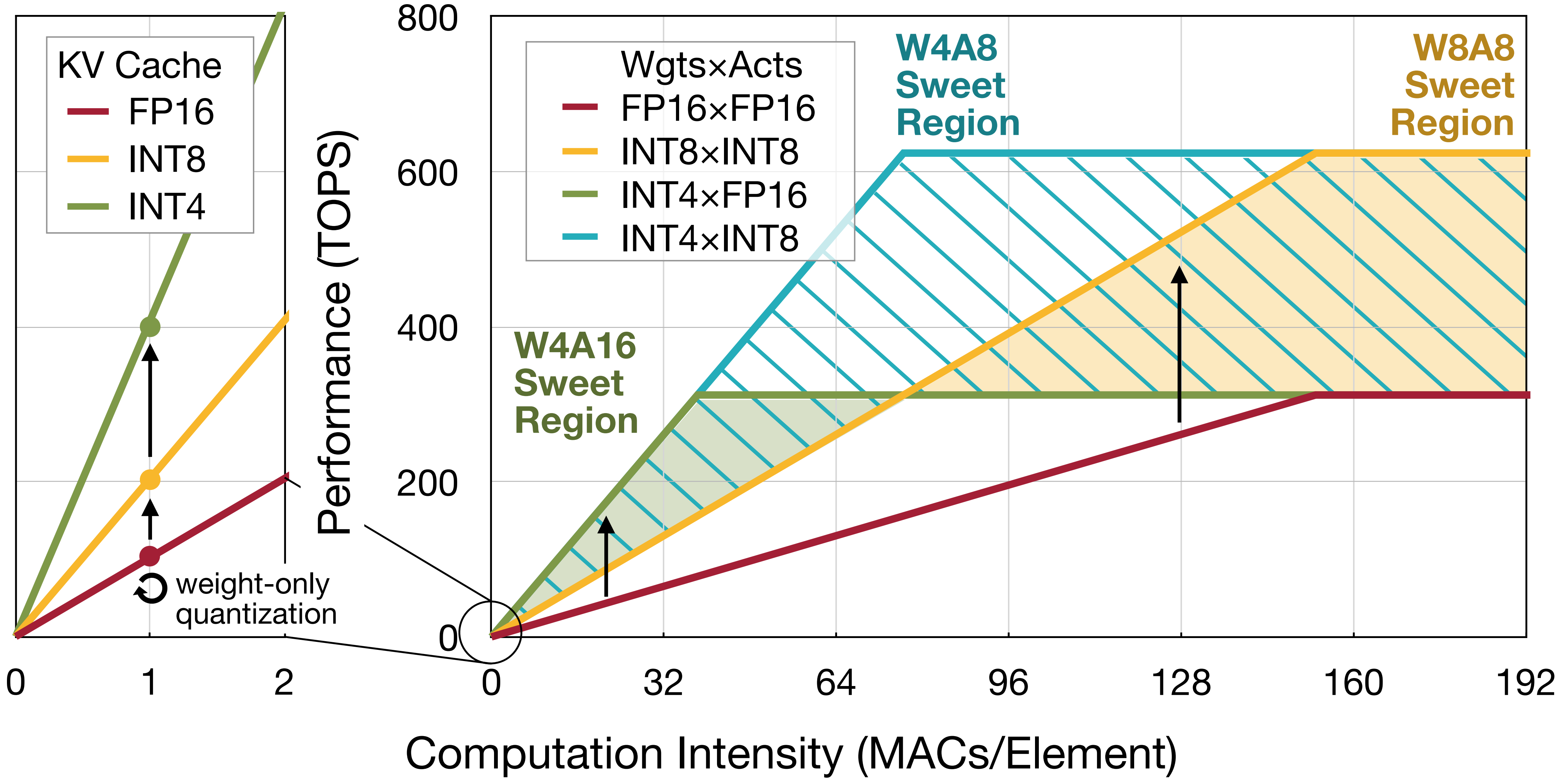


QoQ: W4A8KV4 Quantization

QoQ stands for *quattuor-octo-quattuor*, which represents 4-8-4 in Latin.

- QoQ quantization uses
  - 4-bit weights to save memory bandwidth compared to INT8 weights
  - 8-bit activations to improve peak performance compared to FP16 activations
  - 4-bit KV Cache to save both storage and bandwidth compared to INT8 KV Cache.



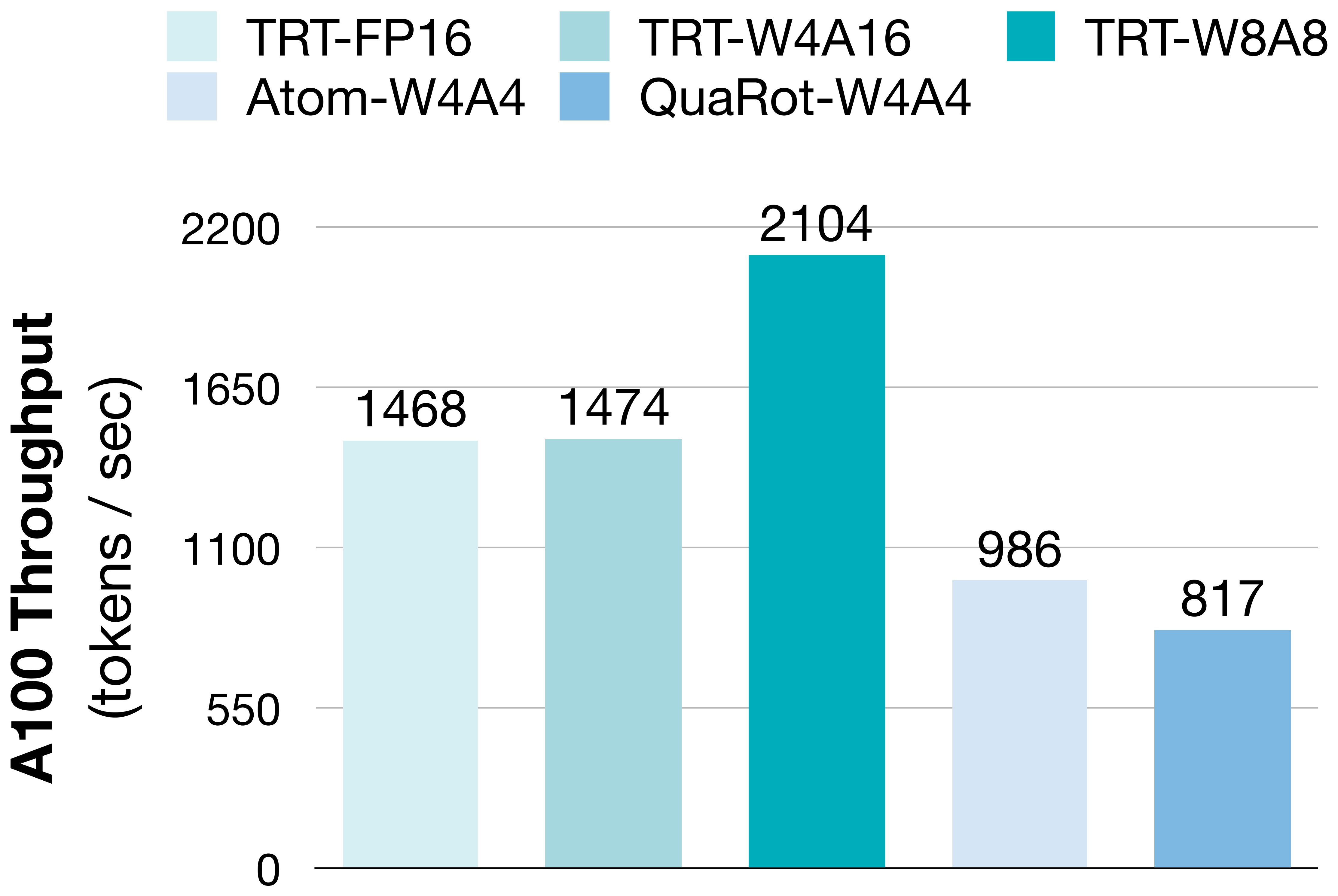


Current status of LLM serving systems

W4A4 quantization leads to severe accuracy loss and even worse efficiency.

- Despite the existence of aggressive W4A4 quantization algorithms, they
  - Bring about significant accuracy loss;
  - Cannot run efficiently on current GPUs.

Wikitext2 Perplexity ↓	Algorithm	Llama-2	
		7B	70B
W8A8	SmoothQuant	5.54	3.36
W4A4	QuaRot	6.19	3.83
W4A4 g128	Atom	**6.12**	**3.73**
W4A8KV4	RTN	6.51	3.90
W4A8KV4 g128	RTN	**5.99**	**3.70**



QoQ: W4A8KV4 Quantization Algorithm for LLM Cloud Serving

Progressive Group Quantization - W4A8

Per-channel INT8 quantization followed by per-group INT4 quantization

❶ $$s^{(0)} = \frac{|w|_{\max}}{q_{\max}} = \frac{|8.10|}{119} = 0.07$$

❷ $$q^{(0)}_{w_{0,0}} = \lfloor \frac{w_{0,0}}{s^{(0)}} \rceil = \lceil \frac{-0.20}{0.07} \rfloor = -3$$

❶ $$s^{(1)} = \lfloor \frac{q^{(0)}_{w_{\max}} - q^{(0)}_{w_{\min}}}{q_{\max} - q_{\min}} \rceil = \lfloor \frac{14 - -16}{15 - 0} \rceil = 2$$

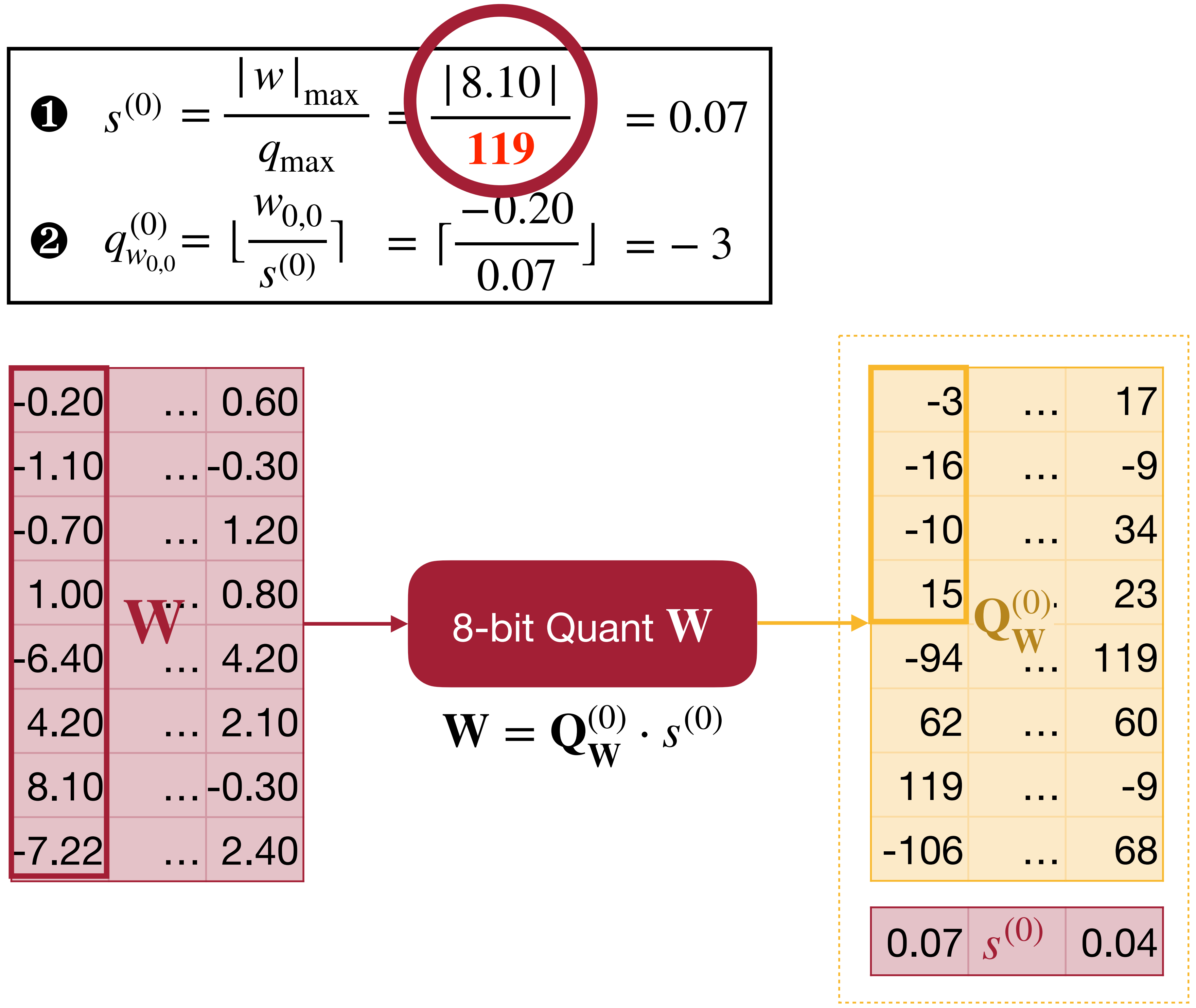
❷ $$z = \lfloor - \frac{q^{(0)}_{w_{\min}}}{s^{(1)}} \rceil = \lfloor - \frac{-16}{2} \rceil = 8$$

❸ $$q_{w_{0,0}} = \lfloor \frac{q^{(0)}_{w_{0,0}}}{s^{(1)}} + z \rceil = \lfloor \frac{-3}{2} + 8 \rceil = 7$$

$$\hat{q}^{(0)}_{w_{0,0}} = (q_{w_{0,0}} - z) \cdot s^{(1)} = (7 - 8) \cdot 2 = -2$$

Progressive Group Quantization - W4A8

Protective range [-119, 119] avoids dequantization overflow.



- We reconsider the dequantization process.

$$\hat{w}_{s8} = \lceil \frac{w_{s8}}{s_{u8}} \rfloor \cdot s_{u8} \leq w_{s4} + \frac{1}{2} s_{u8}$$

- Since

$$s_{u8} = \frac{w_{s8_{max}} - w_{s8_{min}}}{q_{u4_{max}} - q_{u4_{min}}} \leq \frac{q_{s8_{max}} - q_{s8_{min}}}{q_{u4_{max}} - q_{u4_{min}}} = \frac{127 - (-127)}{15 - 0} = 17$$

- to have

$$\hat{w}_{s8} \leq 127$$

- we only need

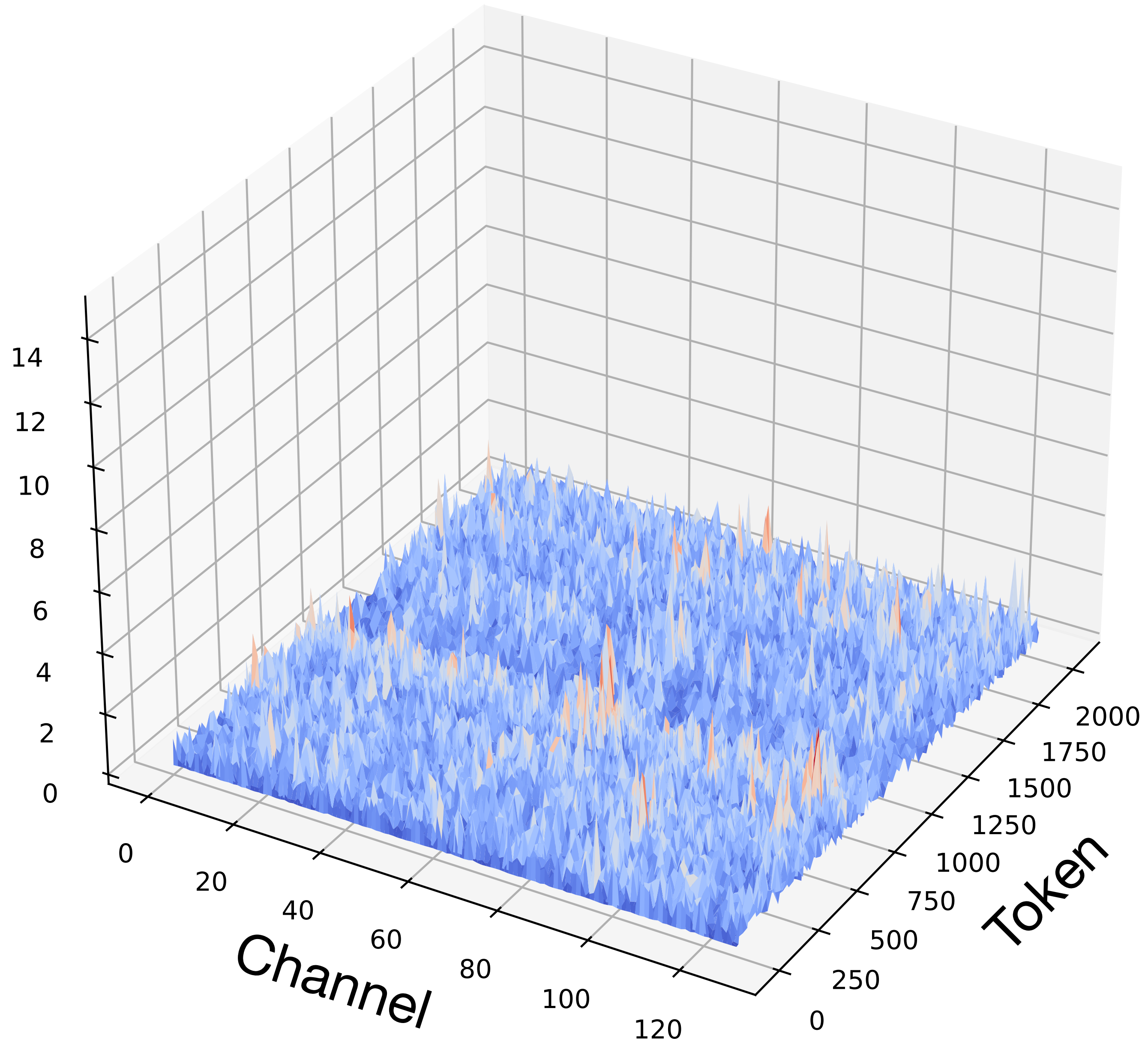
$$w_{s8} + \frac{1}{2} s_{u8} \leq 127$$

- which is

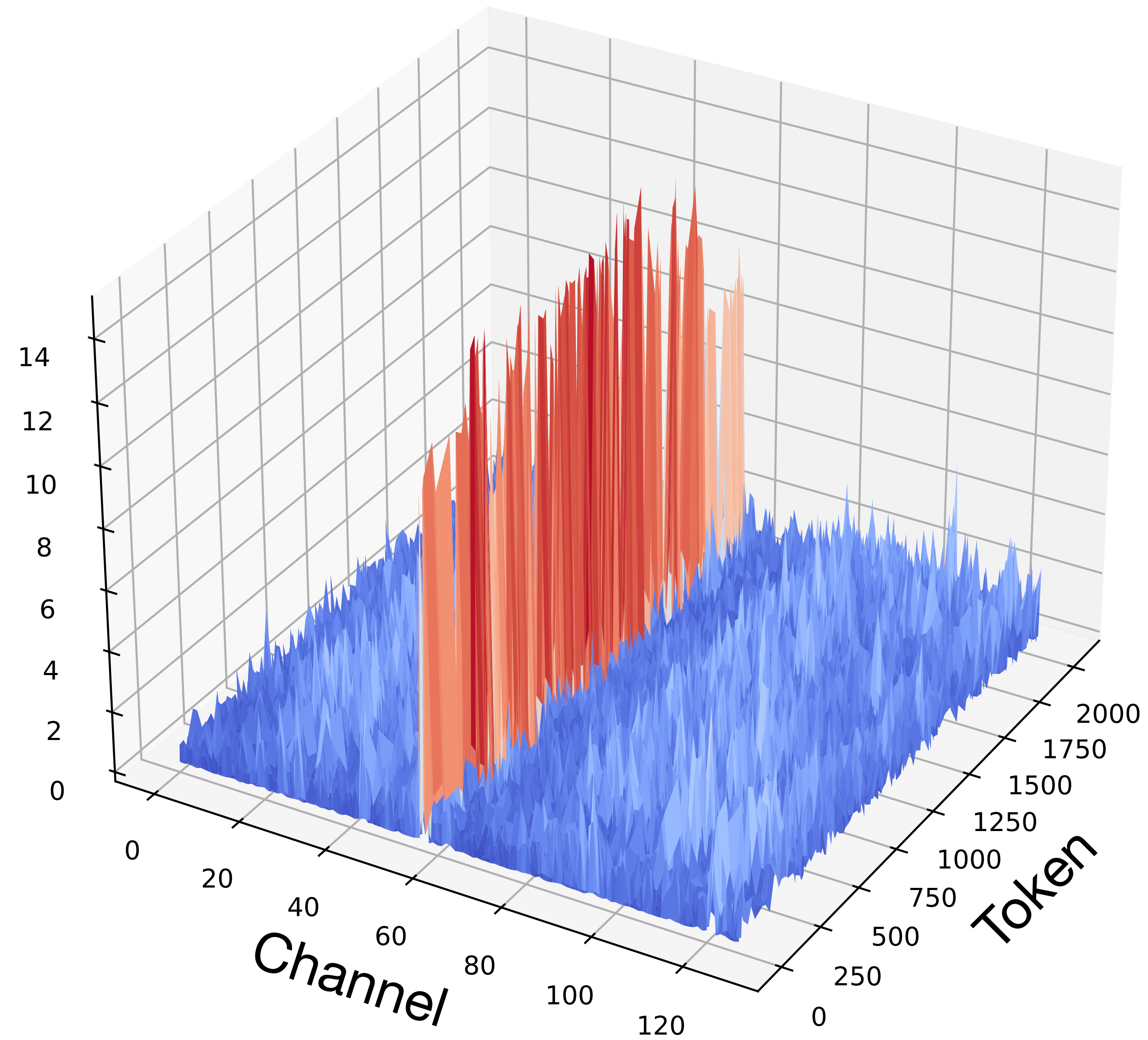
$$w_{s8} \leq 127 - \frac{1}{2} s_{u8} \leq 127 - \frac{1}{2} \cdot 17 = 119.5$$

SmoothAttention - KV4

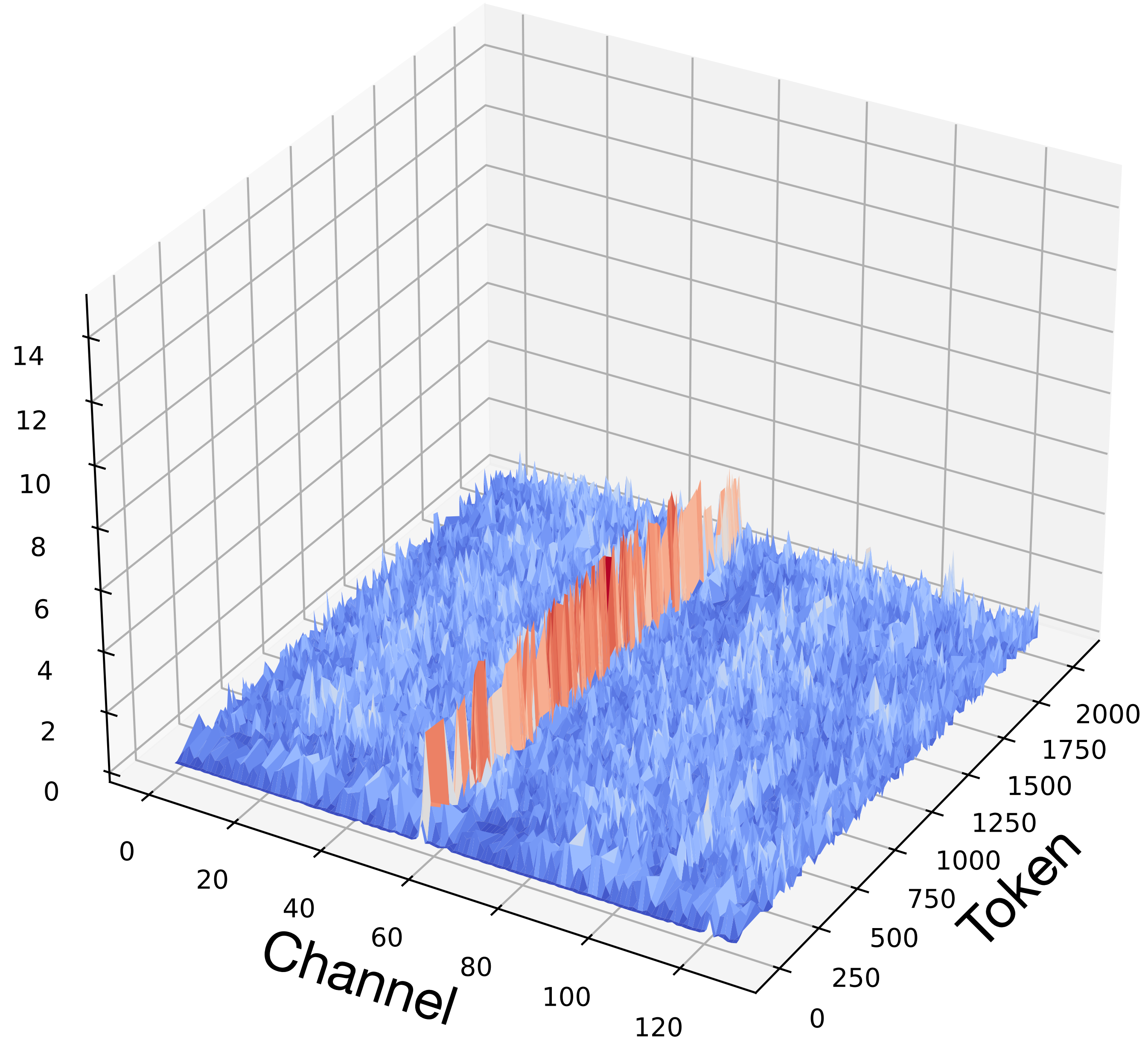
Migrate quantization difficulty from K cache to Q matrix



Layer 24 Values



Layer 24 post-RoPE Keys (Original)



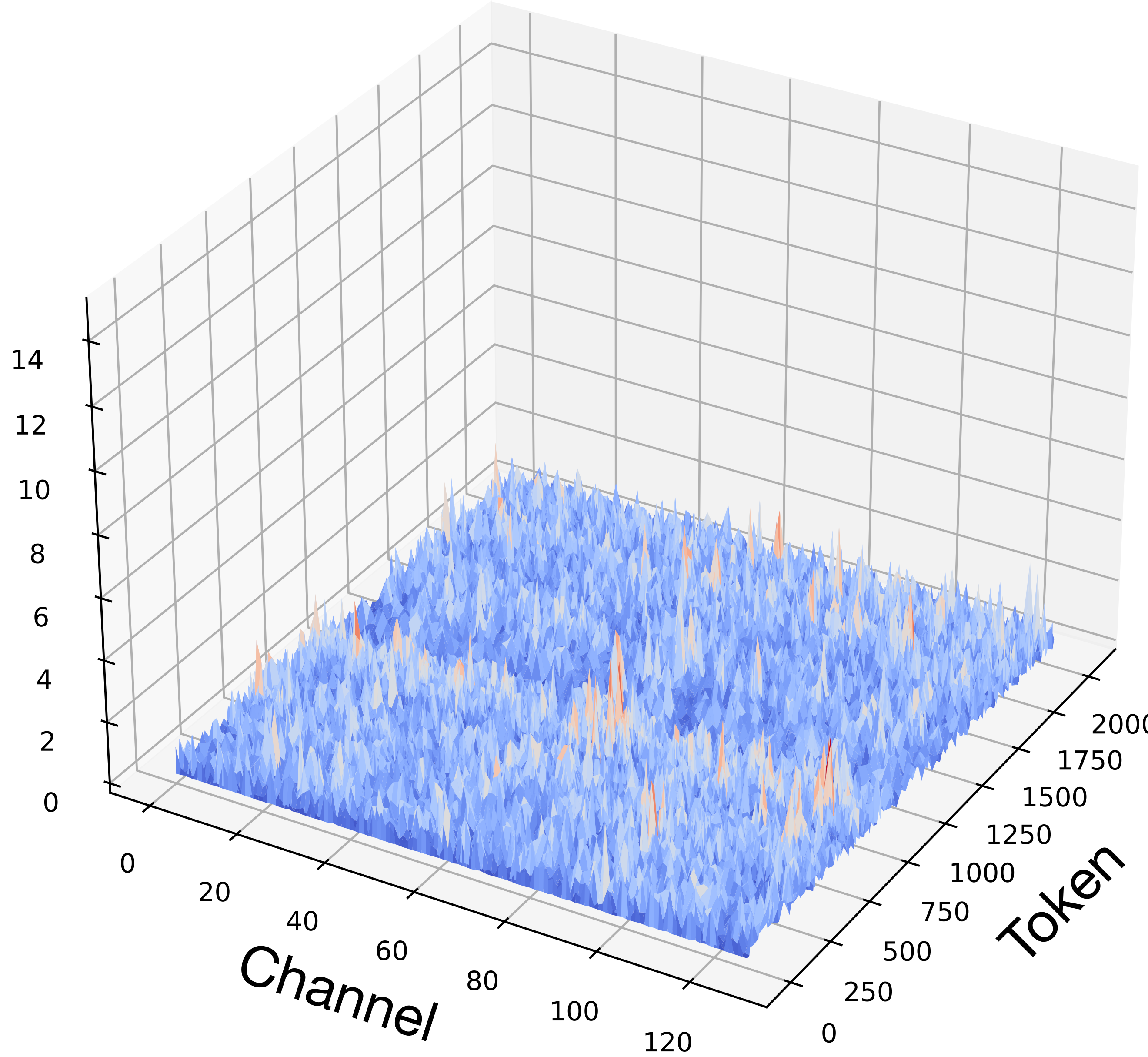
Layer 24 post-RoPE Keys (SmoothAttention)

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

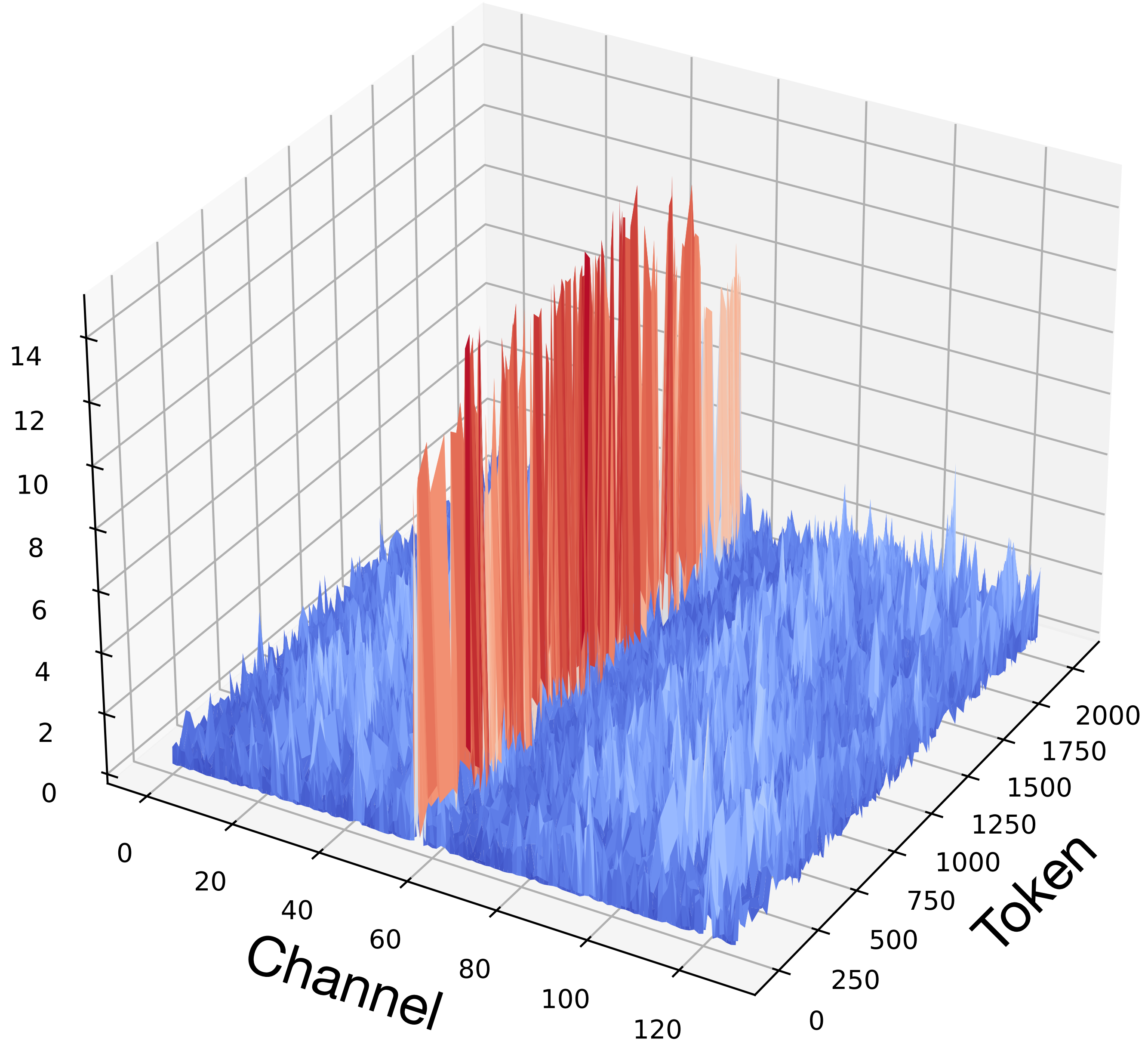
Both Q and K are activations.

SmoothAttention - KV4

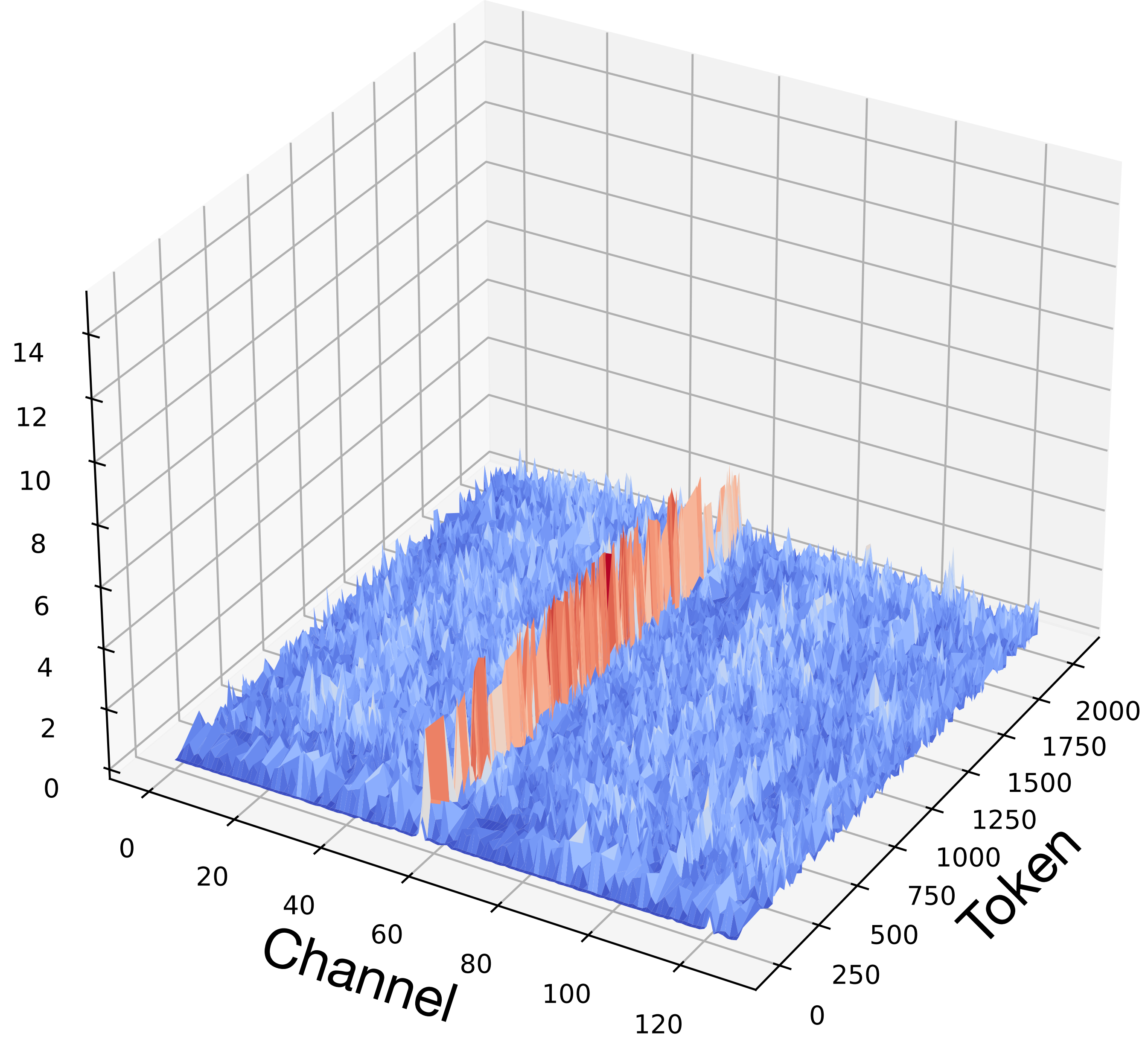
Migrate quantization difficulty from K cache to Q matrix



Layer 24 Values



Layer 24 post-RoPE Keys (Original)



Layer 24 post-RoPE Keys (SmoothAttention)

FP16

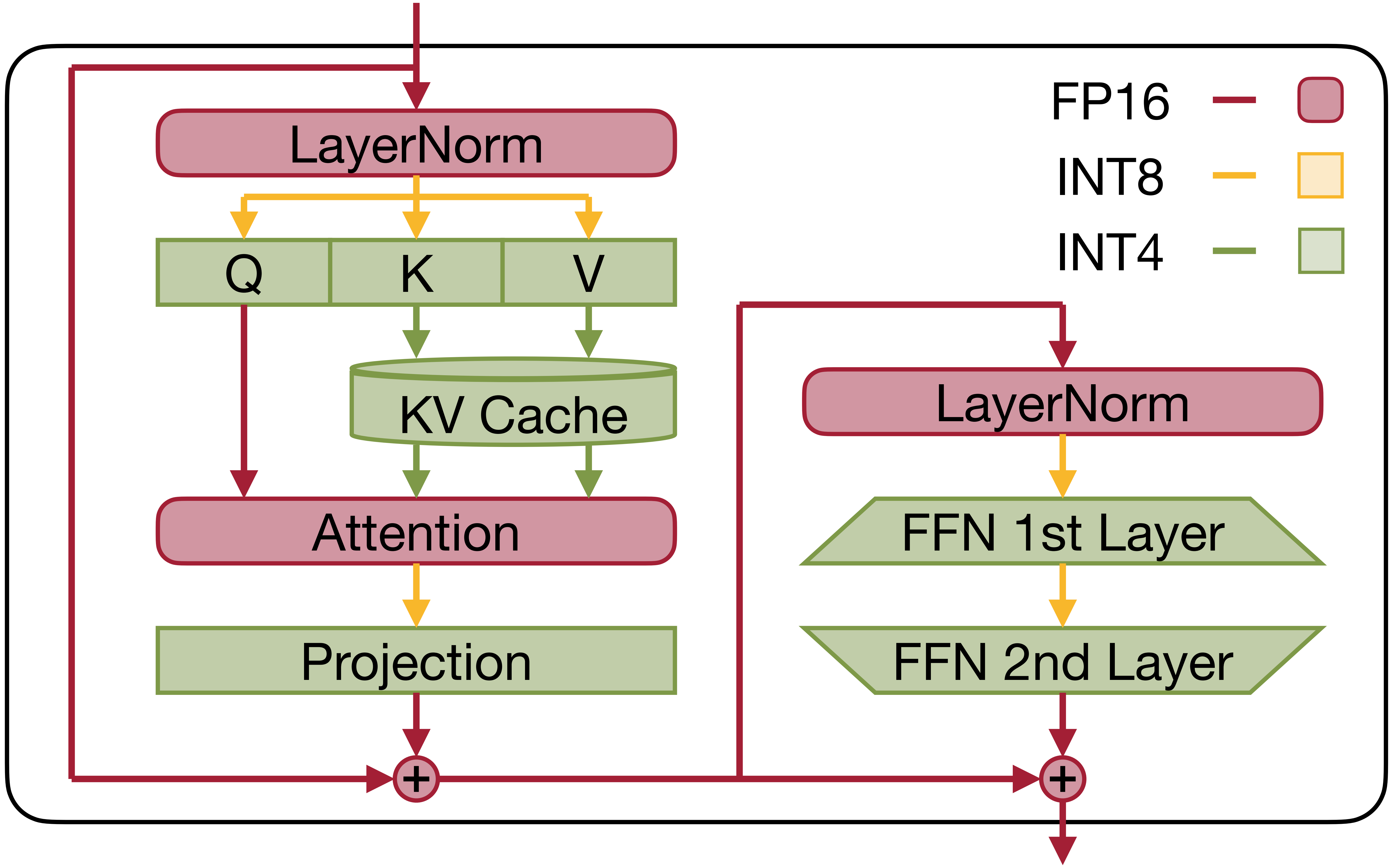
$$\mathbf{Z} = \left(\boxed{\mathbf{Q}}\mathbf{\Lambda}\right) \cdot \left(\boxed{\mathbf{K}}\mathbf{\Lambda}^{-1}\right)^T, \quad \mathbf{\Lambda} = \mathrm{diag}\left(\lambda\right)$$

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

Both Q and K are activations.

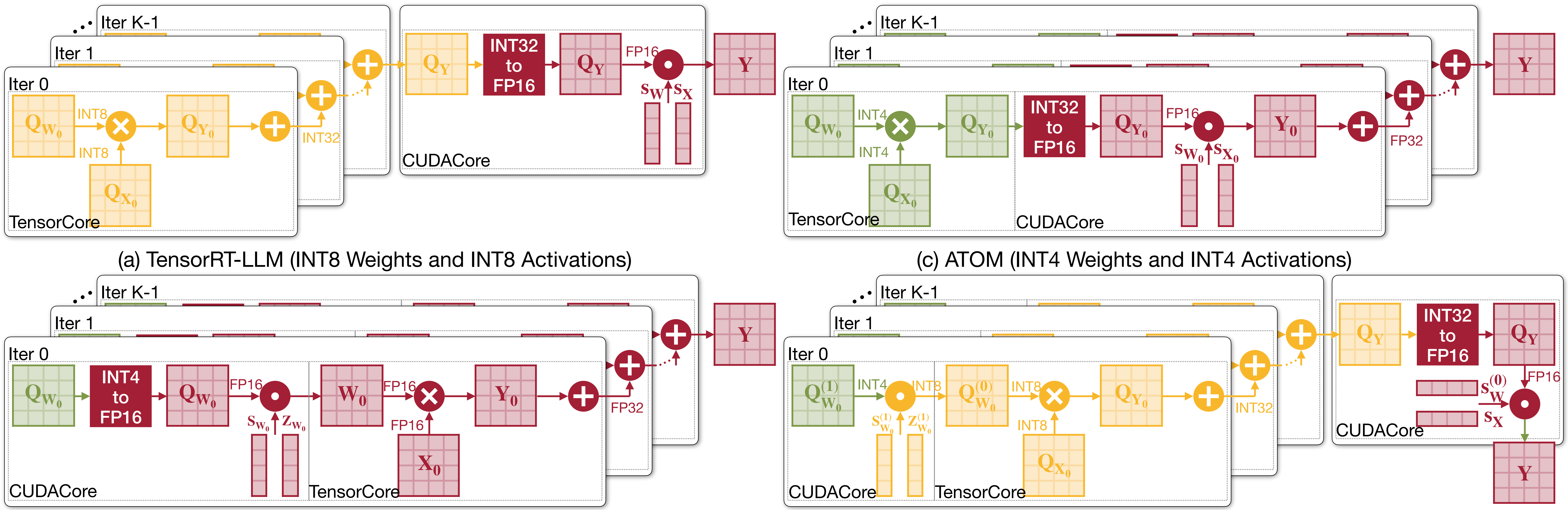
QServe: Efficient LLM Serving System with W4A8KV4 Quantization

QServe System Overview



Quantized GEMM on GPUs

Overhead in the GEMM main loop should be avoided.



(a) TensorRT-LLM (INT8 Weights and INT8 Activations)

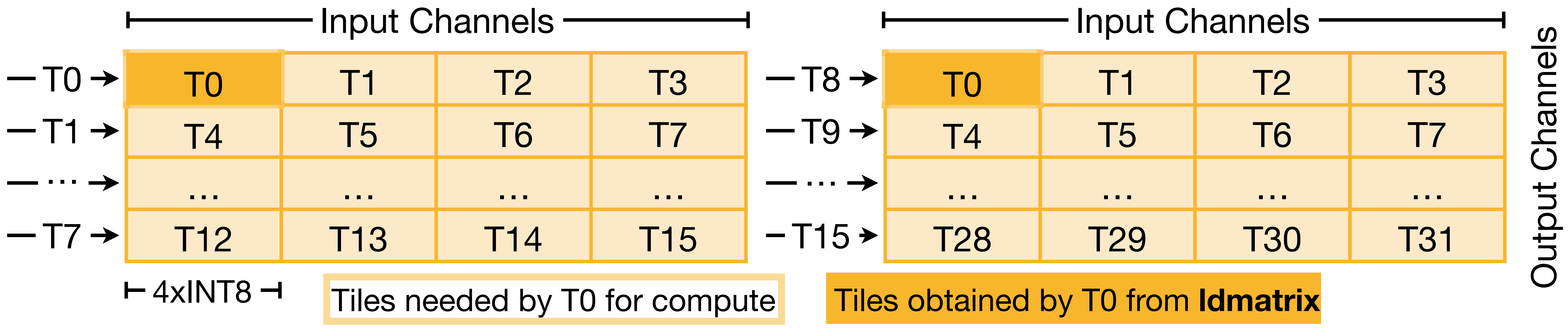
(c) ATOM (INT4 Weights and INT4 Activations)

(b) TensorRT-LLM (INT4 Weights and FP16 Activations)

(d) Ours (INT4 Weights and INT8 Activations)

Compute-Aware Weight Reorder

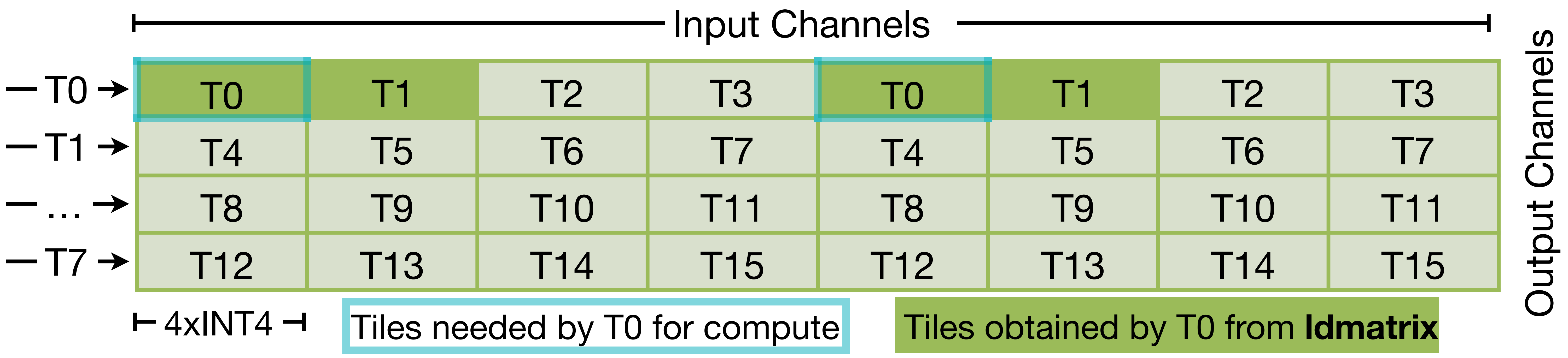
Reducing pointer arithmetics overhead through offline weight reordering



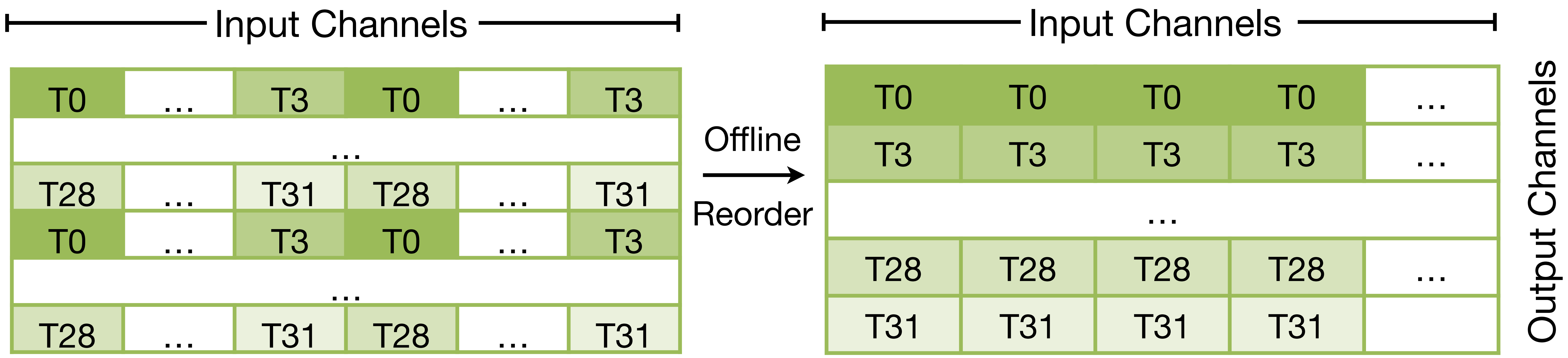
(a) The ldmatrix instruction ensures that each thread gets what it needs for compute in W8A8 GEMM

Compute-Aware Weight Reorder

Reducing pointer arithmetics overhead through offline weight reordering



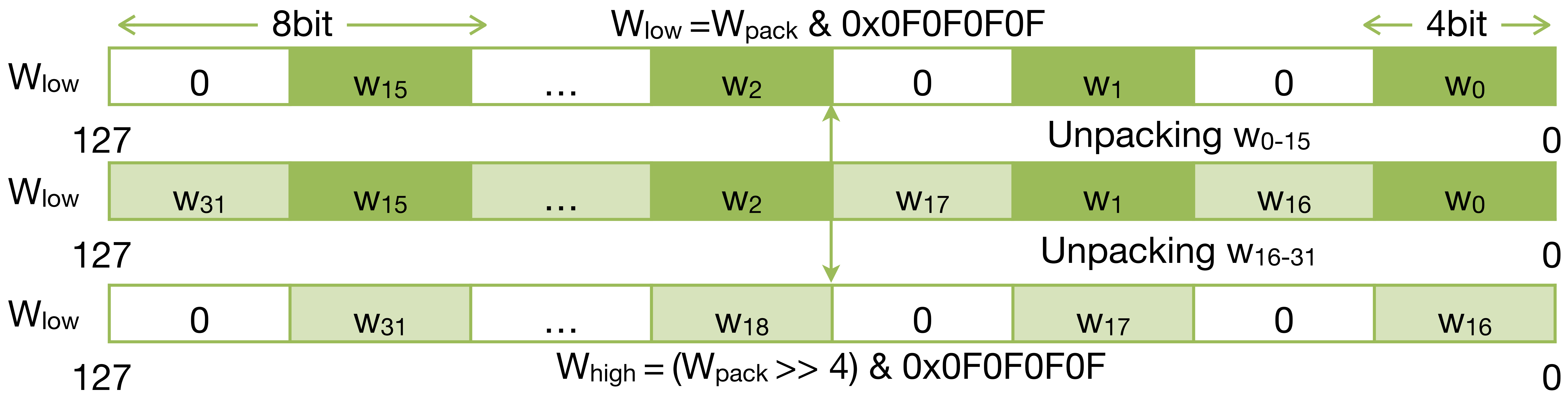
(b) However, the ldmatrix instruction fails for W4A8 GEMM due to **storage-compute mismatch**



(c) Our solution: **compute-aware weight reorder**

Efficient Dequantization with Reg-Level Parallelism

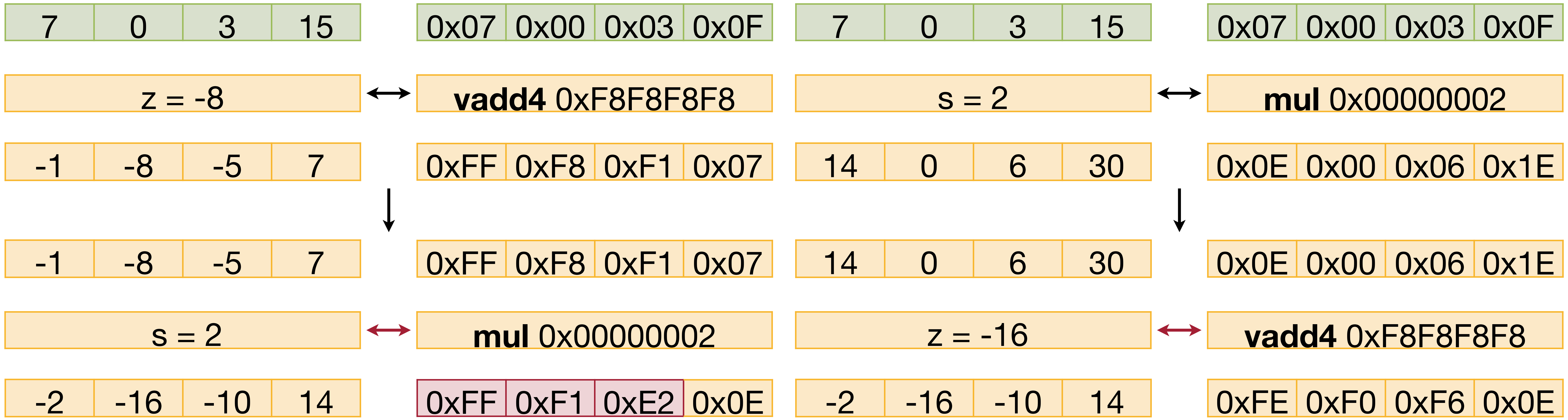
Step 1. Dequantization from UINT4 to UINT8



Reorder every 32 weights to **minimize logical instructions**: 3 instructions to dequantize 32 numbers.

Efficient Dequantization with Reg-Level Parallelism

Step 2. Dequantization from UINT8 to SINT8 (apply zero points and scales)



Subtraction **before** multiplication

(a) **Overflow** fails register level parallelism

Subtraction **after** multiplication

(b) **Progressive Quantization** guarantees RLP

Subtraction after multiplication is more efficient than subtraction before multiplication.

Making Fused Attention Memory Bound

Theoretical computation intensity = 1 does not hold in fused implementation

Technique	Latency (ms)	Speedup over TRTLLM-KV8
TRTLLM-KV8	0.42	1.0x
Naive KV4-g128	0.48	**0.88x**
+Efficient INT4-to-FP16 dequantization	0.44	0.95x
+Control flow simplification	0.39	1.08x
+QK product in half2	0.36	1.17x
+Prefetch scales/zeros	0.32	1.31x
+Tuning tiling parameters	0.30	1.40x
+SV product in half2	0.28	1.50x

Low-bit KV quantization does not speed up decoding stage attention automatically.

QServe backend brings fused attention back to the memory bound region and realizes 1.5x speedup.

End-to-end Throughput Evaluation Results

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

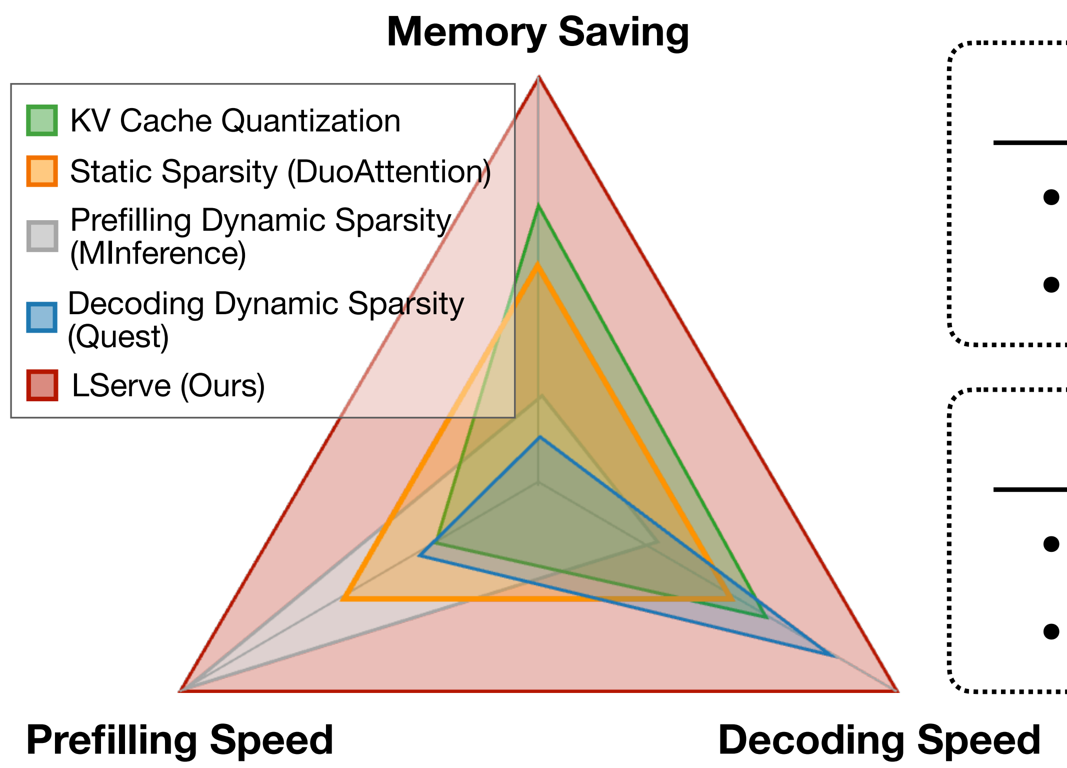
QServe outperforms the state-of-the-art W4A4

Wikitext2 Perplexity ↓		Llama-2			Llama			Mistral	Mixtral	Yi
Precision	Algorithm	7B	13B	70B	7B	13B	30B	7B	8x7B	34B
W8A8	SmoothQuant	5.54	4.95	3.36	5.73	5.13	4.23	5.29	3.89	4.69
W4A16 g128	AWQ	5.60 (+0.06)	4.97 (+0.02)	3.41 (+0.05)	5.78 (+0.05)	5.19 (+0.06)	4.21 (-0.02)	5.37 (+0.08)	4.02 (+0.13)	4.67 (-0.02)
W4A4	QuaRot	6.19 (+0.65)	5.45 (+0.50)	3.83 (+0.47)	6.34 (+0.61)	5.58 (+0.45)	4.64 (+0.41)	5.77 (+0.48)	NaN	NaN
W4A4 g128	Atom	6.12 (+0.58)	5.31 (+0.36)	3.73 (+0.37)	6.25 (+0.52)	5.52 (+0.39)	4.61 (+0.38)	5.76 (+0.47)	4.48 (+0.59)	4.97 (+0.28)
W4A8KV4	QServe	**5.75** (+0.18)	**5.12** (+0.17)	**3.52** (+0.16)	**5.93** (+0.20)	**5.28** (+0.15)	**4.34** (+0.11)	**5.45** (+0.16)	**4.18** (+0.29)	**4.74** (+0.05)
W4A8KV4 g128	QServe	**5.70** (+0.13)	**5.08** (+0.13)	**3.47** (+0.11)	**5.89** (+0.16)	**5.25** (+0.12)	**4.28** (+0.05)	**5.42** (+0.13)	**4.14** (+0.25)	**4.76** (+0.07)

Next Step: Long-context Serving with Sparse Attention

LServe: Efficient Long-sequence LLM Serving with ***Unified Sparse Attention***

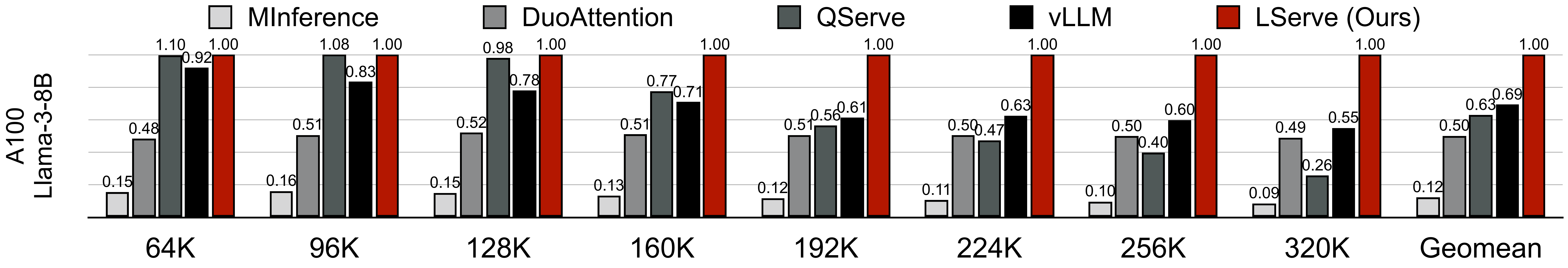
- Unified static and dynamic sparse attention
- Hardware-aware sparse paging system design
- LServe accelerates long-sequence decoding by 1.3-2.1x over vLLM
- Code release:
  - https://github.com/mit-han-lab/omniserve



Sparse Attention Unification
- Unified Framework for Sparse Attention
- Hybrid Static & Dynamic Sparsity

System-Algorithm Co-optimization
- Hierarchical Paging System
- Reusable Page Selector

LServe: Efficient Long-Sequence LLM Serving



LServe: Efficient Long-sequence LLM Serving with Unified Sparse Attention [Yang et al, MLSys 2025]

Thank you!

https://hanlab.mit.edu/projects/qserve

https://github.com/mit-han-lab/omniserve