Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization

Jungi Lee, Wonbeom Lee, Jaewoong Sim

ISCA (2024)

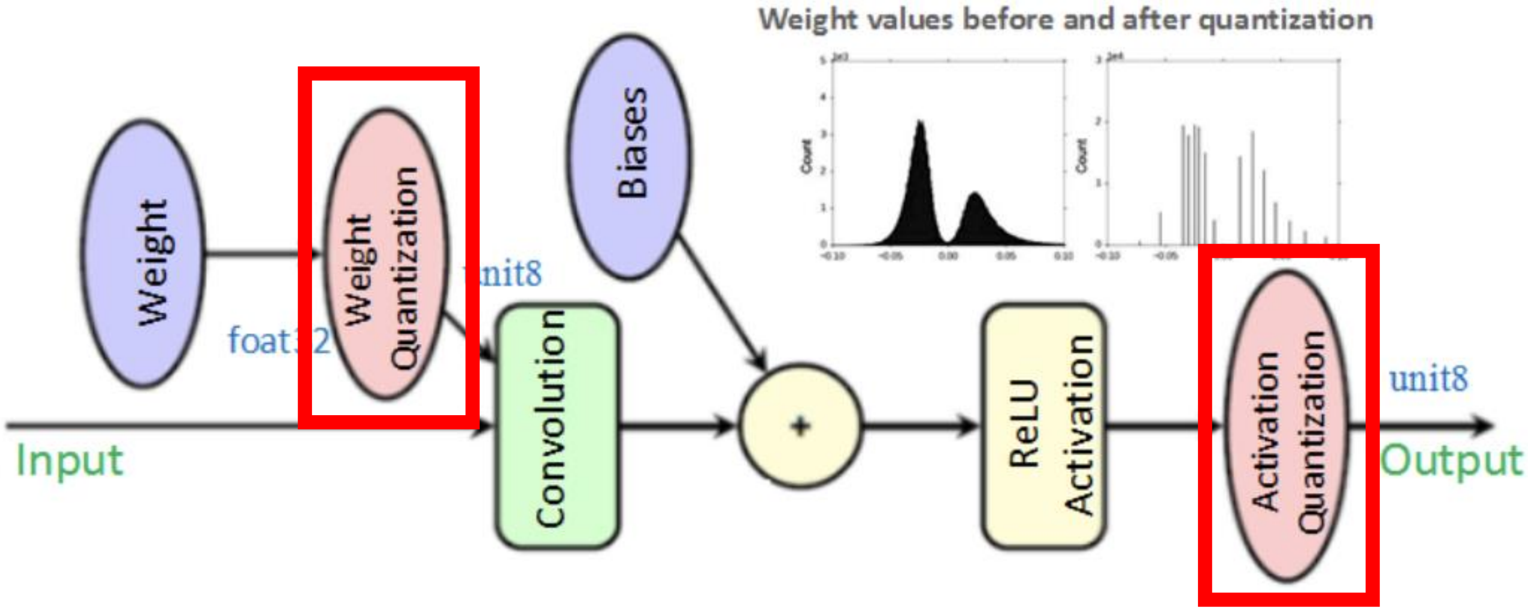
Seoyoon Kim

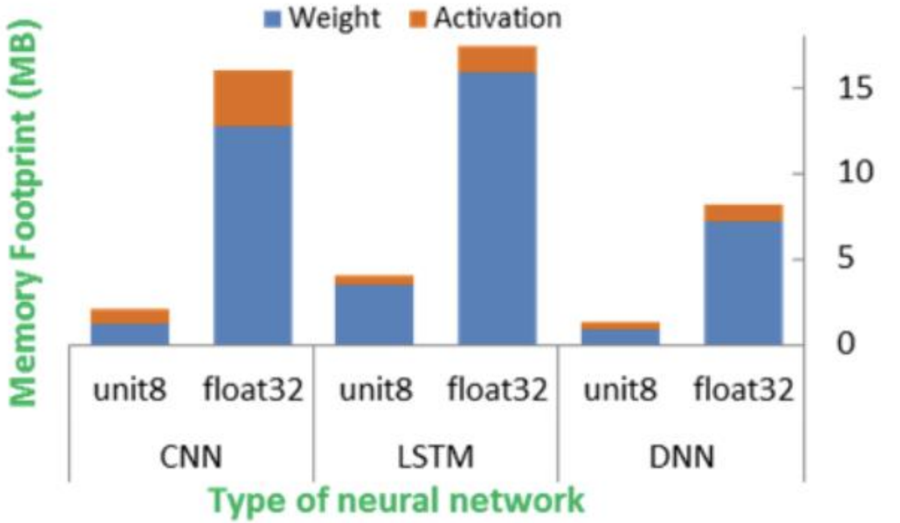
2024-07-19

Contents

Introduction

- ❑ LLM inference = Massive compute & memory resources
- ❑ Quantize *weight* and *activation* into low-bit integers

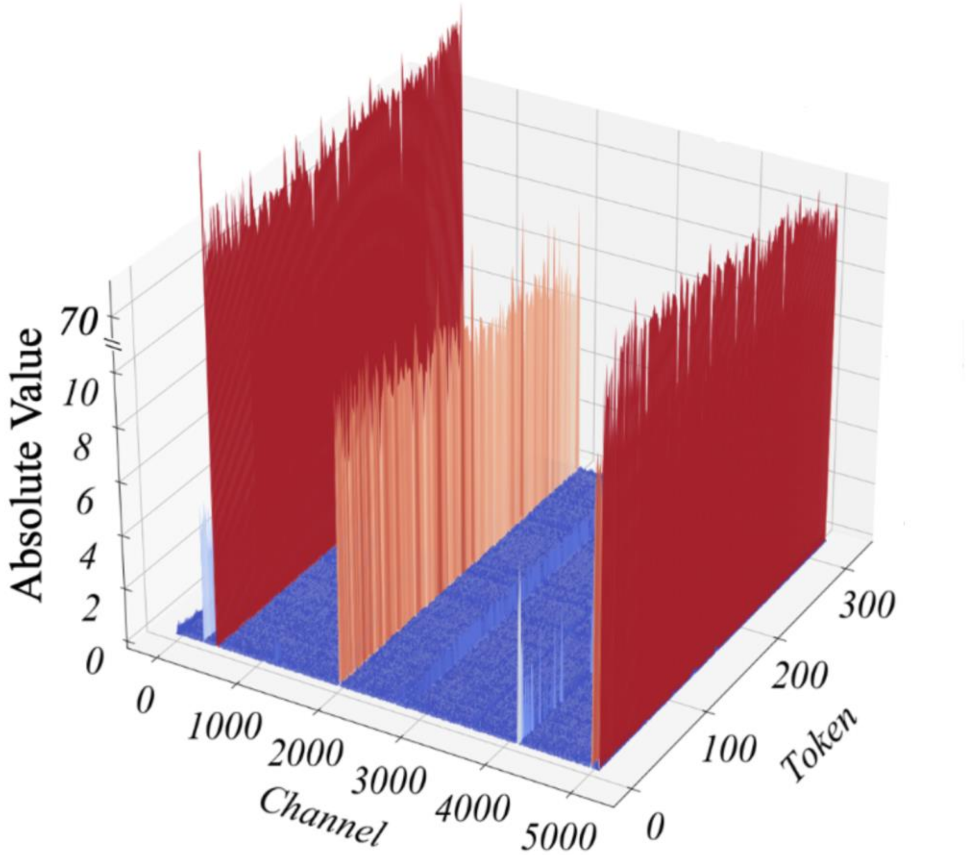




HYPER ACCEL

Introduction

- *Outliers* appear in *activation* (beyond 6.7B parameters)



Activation (Original)

Hard to quantize

HYPER ACCEL

Introduction

❑ Problems of quantizing *activations* in LLMs

1. Software-only works
   - Overhead of complex algorithms
   - Significant quantization loss at ultra low-bit precision

2. Algorithm-hardware co-design
   - Mixed precision / complex compute units
   - Custom / adaptive datatypes

Tender

Introduction

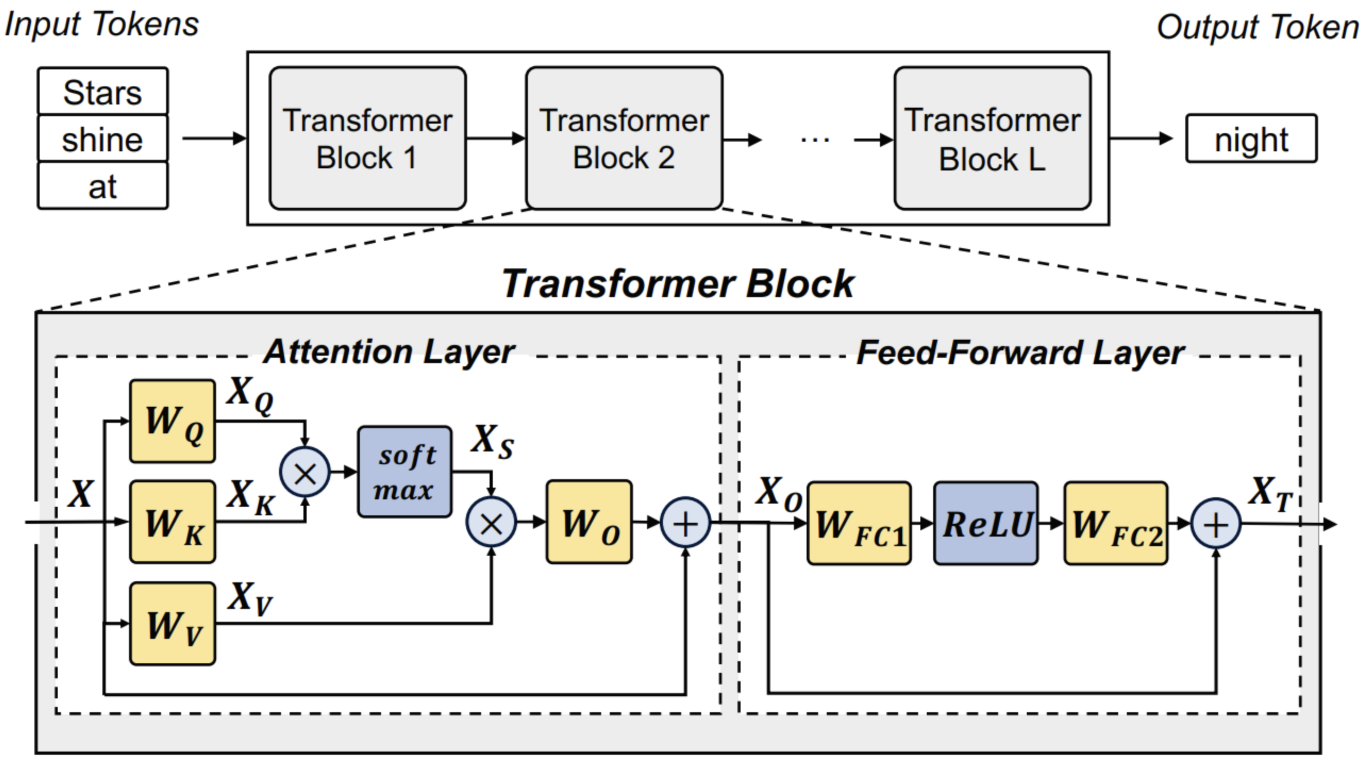
❑ **Tender**

- Decompose *activation* tensor → several *subtensors* (along *channel*)
- Scale factors with *power-of-two* relationships

Shifter Logic → No explicit floating-point operations
Simple Hardware

Background and Motivation

❑ Transformer



$$X_Q = XW_Q; X_K = XW_K; X_V = XW_V$$

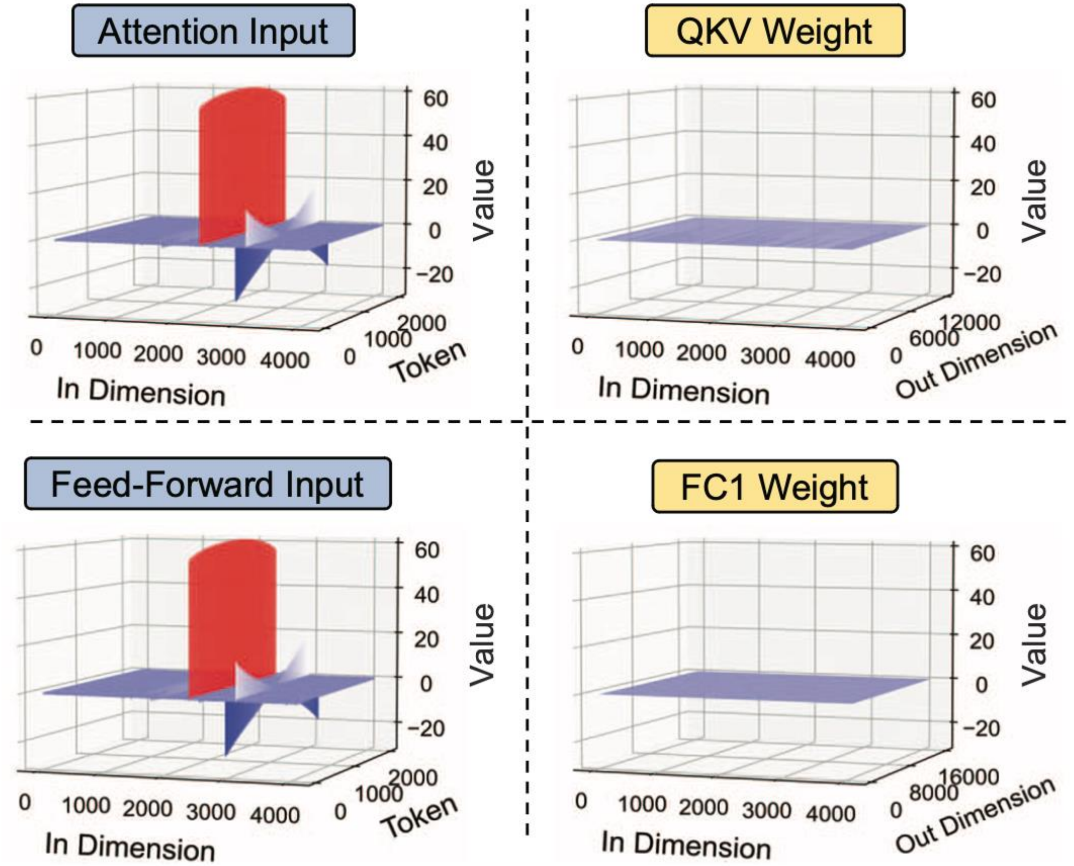
$$X_S = softmax(X_Q X_K^T)$$

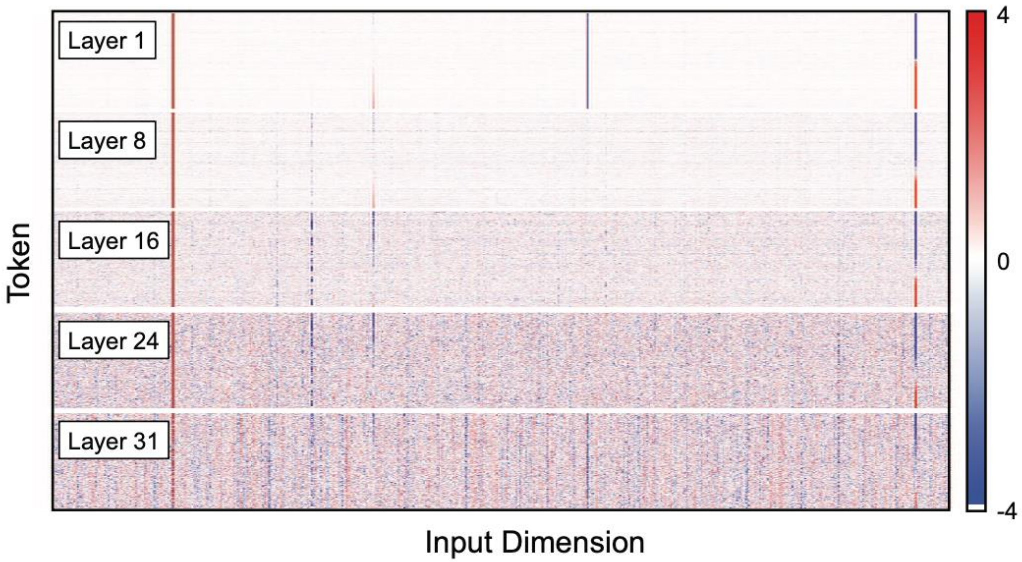
$$X_O = X_S X_V W_O + X$$

$$X_T = ReLU(X_O W_{FC1}) W_{FC2} + X_O$$

Background and Motivation

- ❑ *Outliers* in LLMs
  - Concentrated in fixed channels of activation tensors





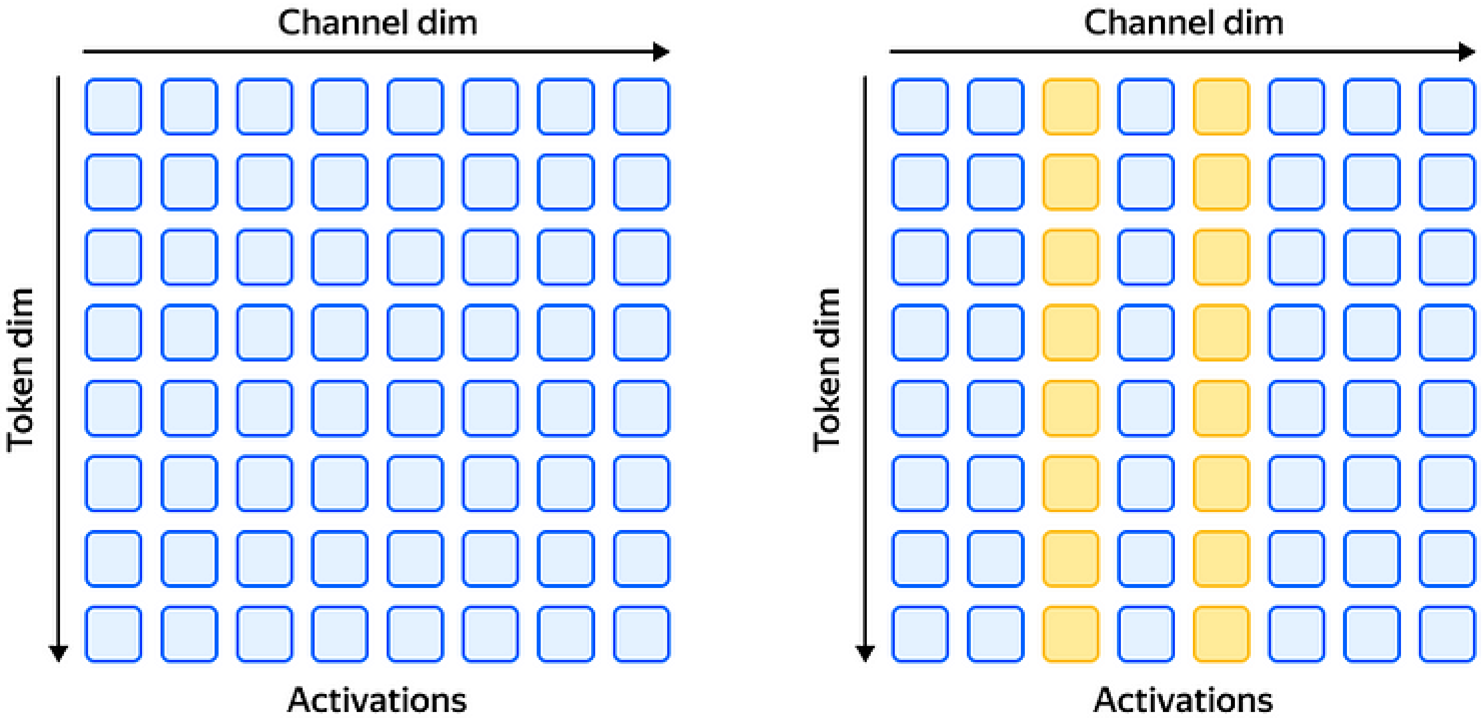
$$s = \frac{x_{max}}{2^{b-1}-1};\quad x_q = round(\frac{x_f}{s})$$

HYPER ACCEL

Background and Motivation

❑ Quantization Granularity

- per-tensor, per-row (= per-token), per-column (= per-channel)



Background and Motivation

❑ Challenges in Quantizing LLMs

Model Performance (perplexity)

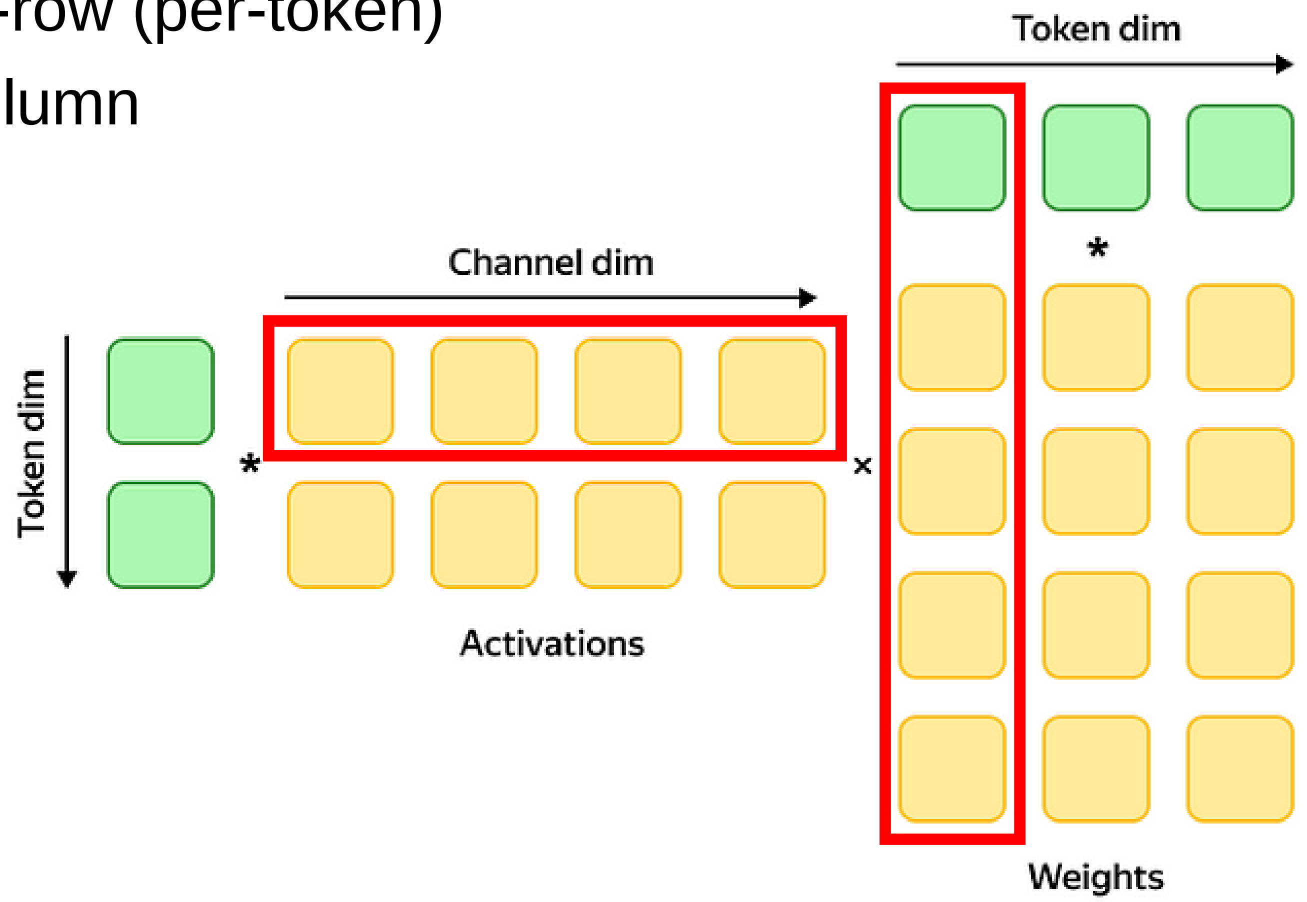
Models	OPT-6.7B	OPT-13B	Llama-2-7B	Llama-2-13B
FP16	10.86	10.13	5.47	4.88
INT8 per-tensor	26.73	4E+3	8.54	51.45
INT8 per-row	20.02	3E+3	5.58	4.94
INT8 per-column	**10.87**	**10.13**	**5.48**	**4.89**
INT4 per-tensor	1E+6	9E+8	4E+4	2E+4
INT4 per-row	1E+6	1E+9	1E+3	5E+3
INT4 per-column	**19.38**	**14.60**	**7.73**	**6.47**

However, per-column quantization poses challenges
since each element needs **scaling** during the reduction operations

Background and Motivation

❑ Quantization Granularity

- Activation: per-row (per-token)
- Weight: per-column



HYPER ACCEL

Background and Motivation

❑ Challenges and Opportunities

$$P_i = \frac{X_i \times W_i}{s_i s_w}, \qquad Y = \sum_{i=1}^{G} (s_i s_w) \cdot P_i$$

P_i: partial sum, Y: final result, s_i, s_w: scale factor

$$A_1 = P_1, \qquad A_{i+1} = A_i \cdot \frac{s_i}{s_{i+1}} + P_{i+1},$$
$$Y = A_G \cdot (s_w s_G)$$

Lower utilization of compute cores due to smaller submatrices and **frequent rescaling**

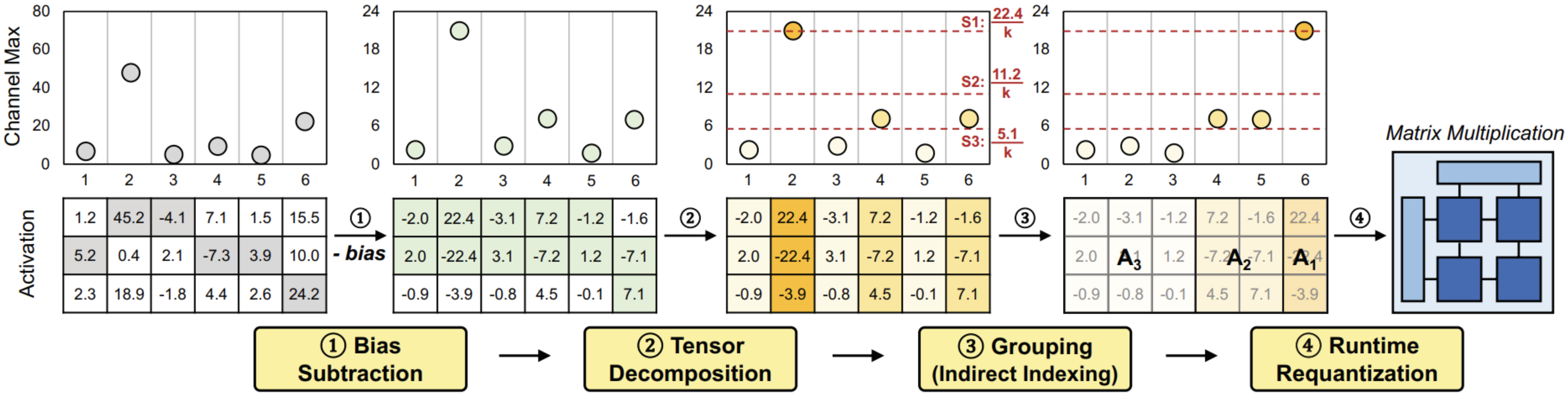
$$rescale\ factor = \frac{s_i}{s_{i+1}} = 2^g$$

→ Group channels with similar ranges to isolate outliers

→ Process partial sums in a specific order & rescale using integer MAC units

Algorithmic Implementation

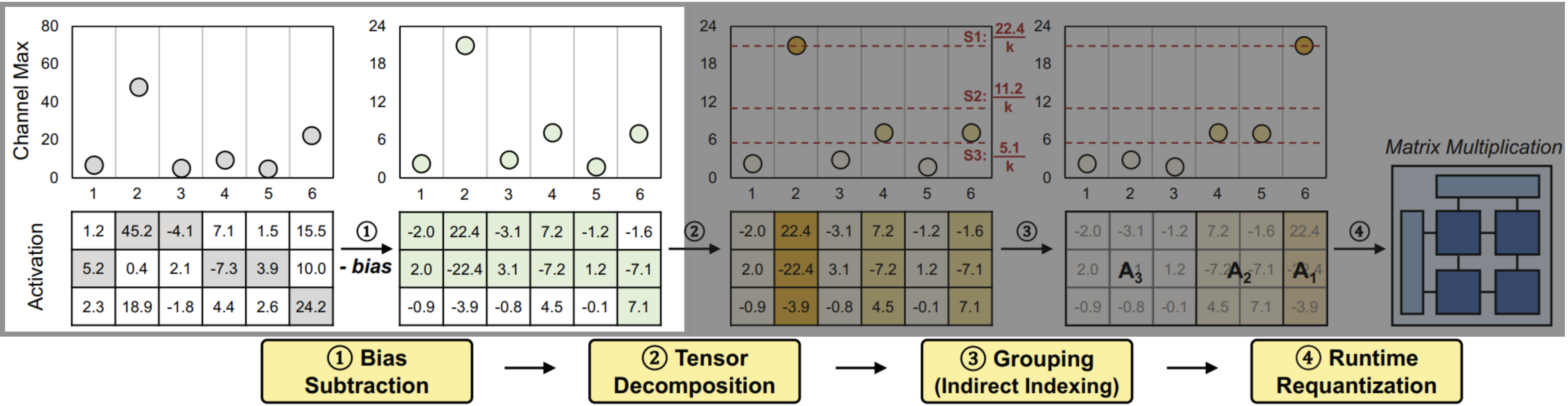
❑ Tender Computation Flow



Algorithmic Implementation

❑ Tensor Computation Flow

1. Bias Subtraction

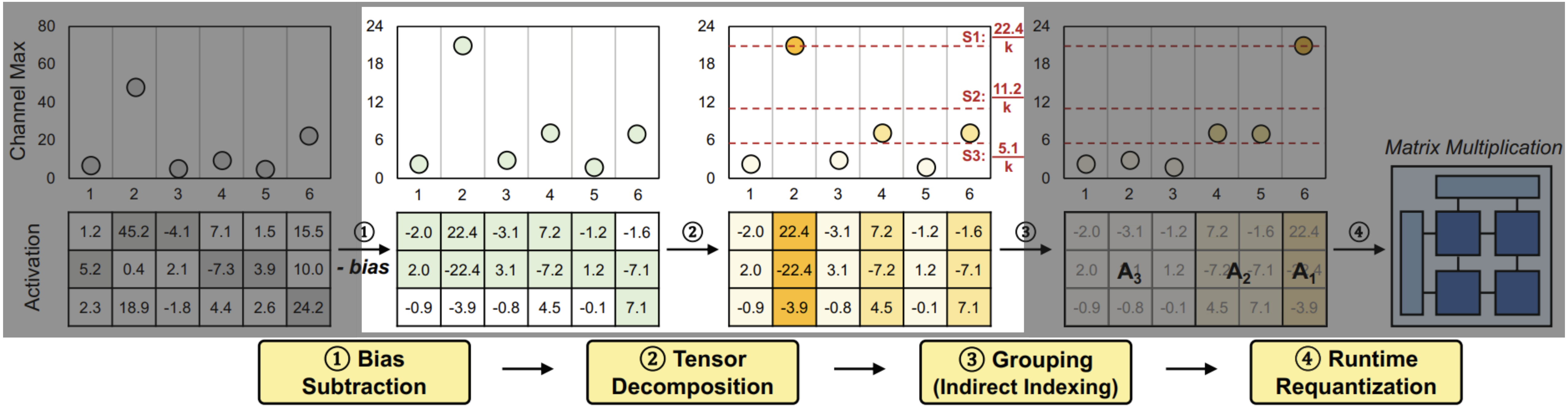


- Subtract *bias* of each channel from the activation tensor
- *bias* = (*max* + *min*) / 2
- Ensure the absolute values of *max* and *min* elements are equal, optimizing the bit usage

Algorithmic Implementation

❑ Tensor Computation Flow

2. Tensor Decomposition



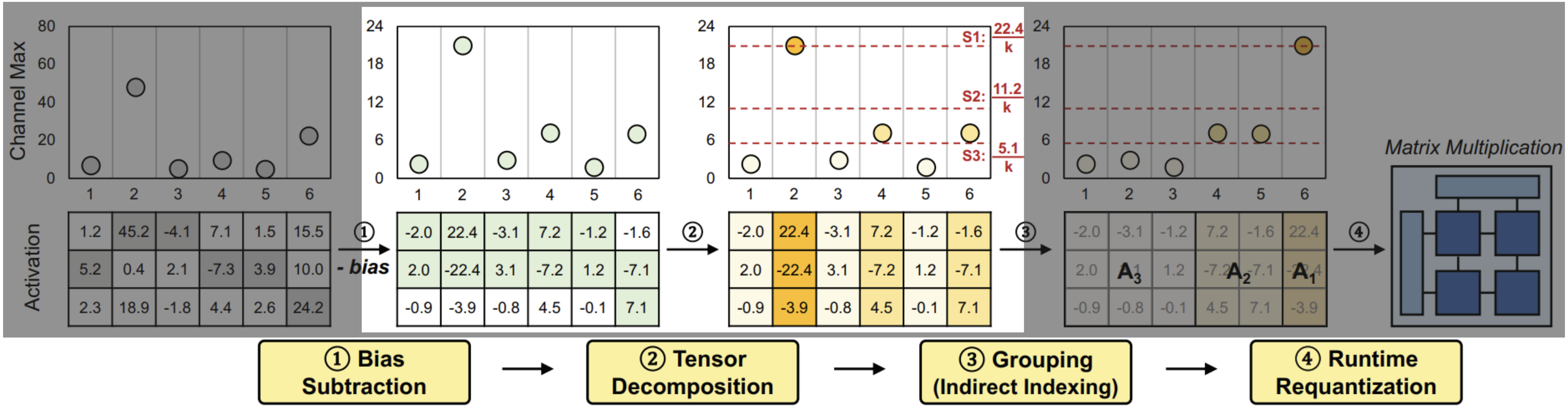
- Compute CMax & Tmax
- Assign i-th channel to group g satisfying: $\boxed{\frac{TMax}{\alpha^{g}} < CMax_i \le \frac{TMax}{\alpha^{g-1}},}\ g = 1,2,\ldots,G$
- Every channel in group g is quantized using the same scale factor: $\boxed{\frac{TMax}{\alpha^{g-1}(2^{b-1}-1)}}$
- Rescaling becomes simple shifting if $\alpha = 2$

HYPER ACCEL

Algorithmic Implementation

❑ Tensor Computation Flow

2. Tensor Decomposition



Why use coarse-grained threshold for large values and fine-grained thresholds for small values?

$$scale\ factor = \frac{TMax}{\alpha^{g-1}(2^{b-1}-1)} \qquad \frac{TMax}{\alpha^{g}} < CMax_i \leq \frac{TMax}{\alpha^{g-1}}$$

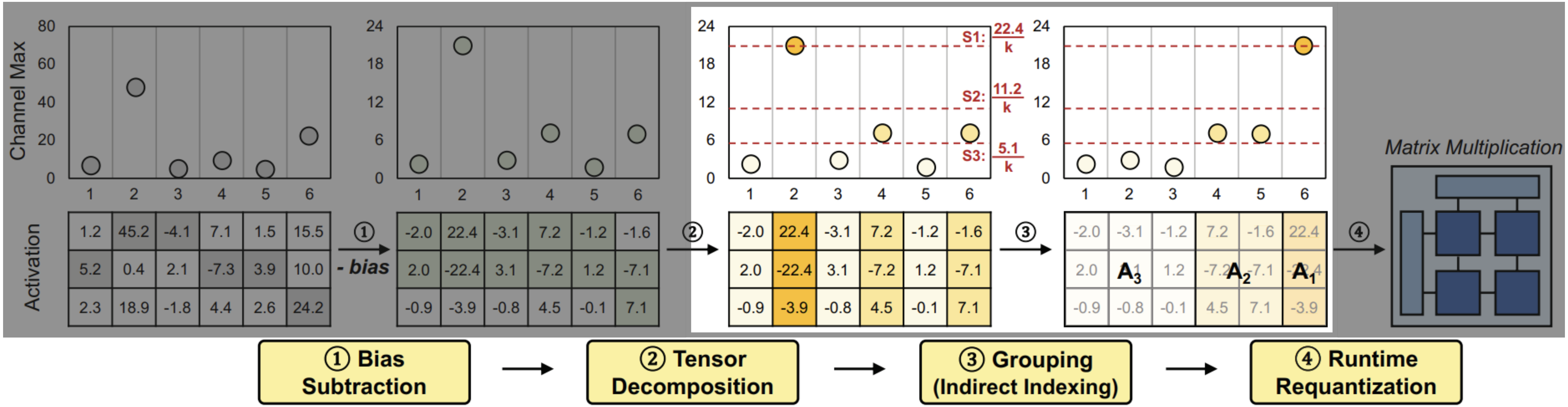
$$Q_{err\_max} = 0.5 \times scale\ factor$$

$$Q_{err} \propto Absolute\ Maximum\ \times Number\ of\ Channels$$

Algorithmic Implementation

❑ Tensor Computation Flow

3. Grouping

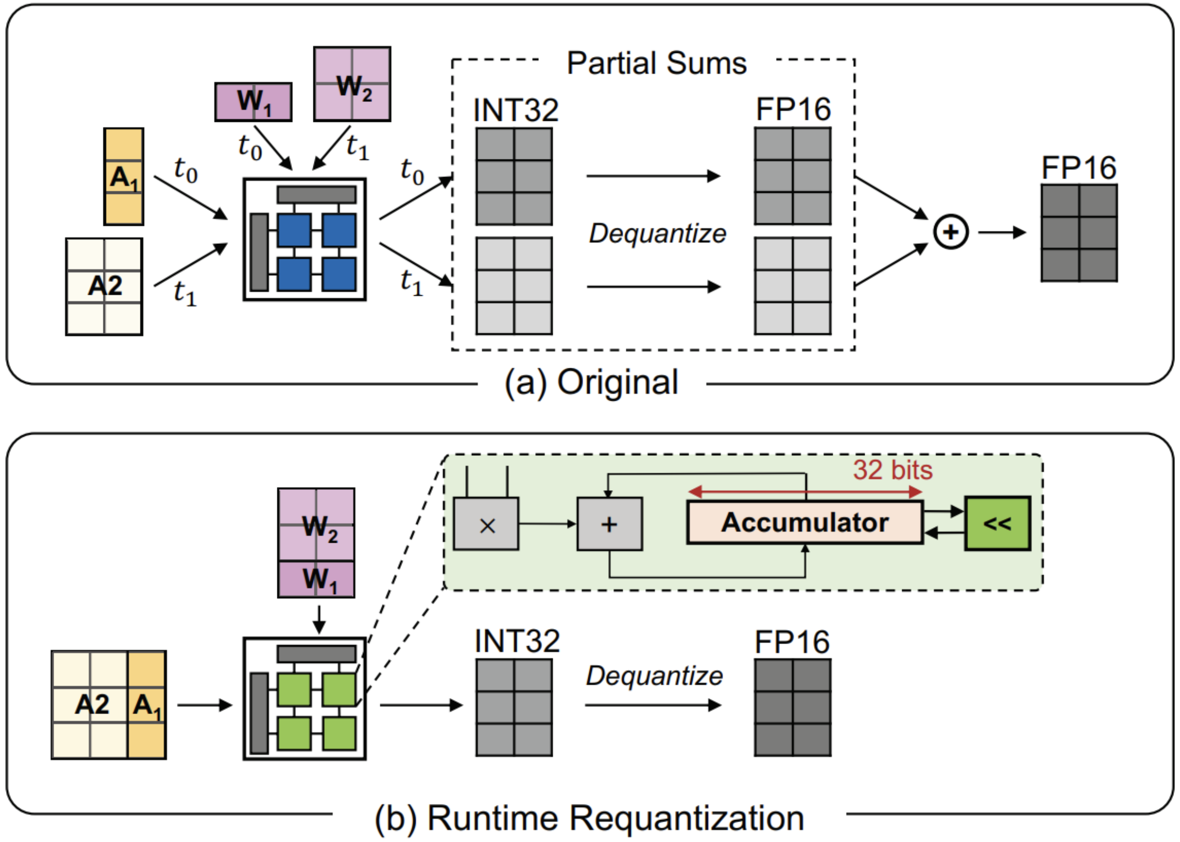


- Classify each channel into several groups using indirect indexing

Algorithmic Implementation

❑ Tensor Computation Flow

4. Runtime Requantization

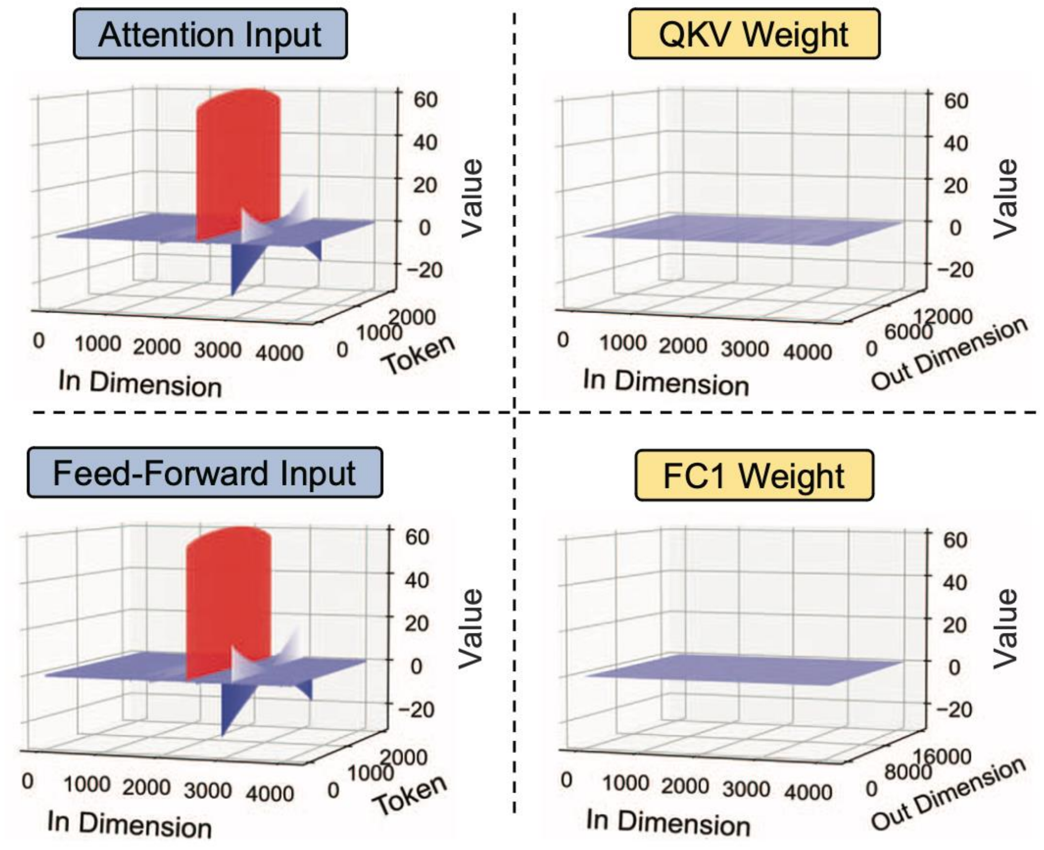


- Before computing the next group, **Shifter** shifts the accumulated integer by 1-bit

Algorithmic Implementation

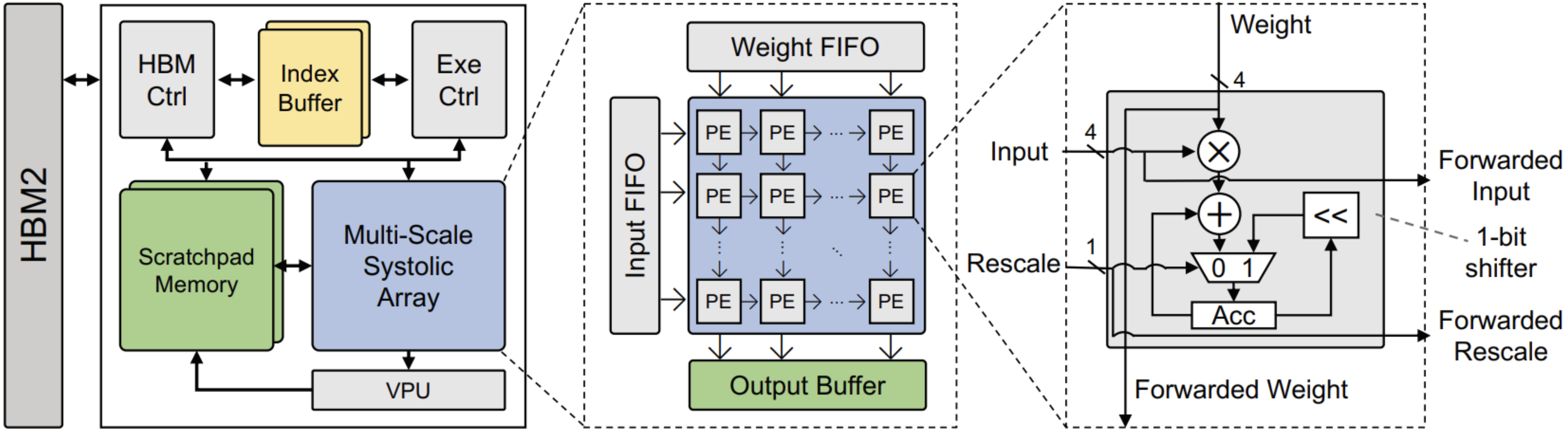
❑ Optimization

- Not only *inter-channel* variances but also *intra-channel* variances
- **Row chunking** (No additional complexity): Divide the rows of the activation tensor & calibrate biases and scale factors offline
- Chunk size = 256 (underutilization vs fine-grained row grouping)



Hardware Architecture

❑ 1. Overview of Tender Architecture

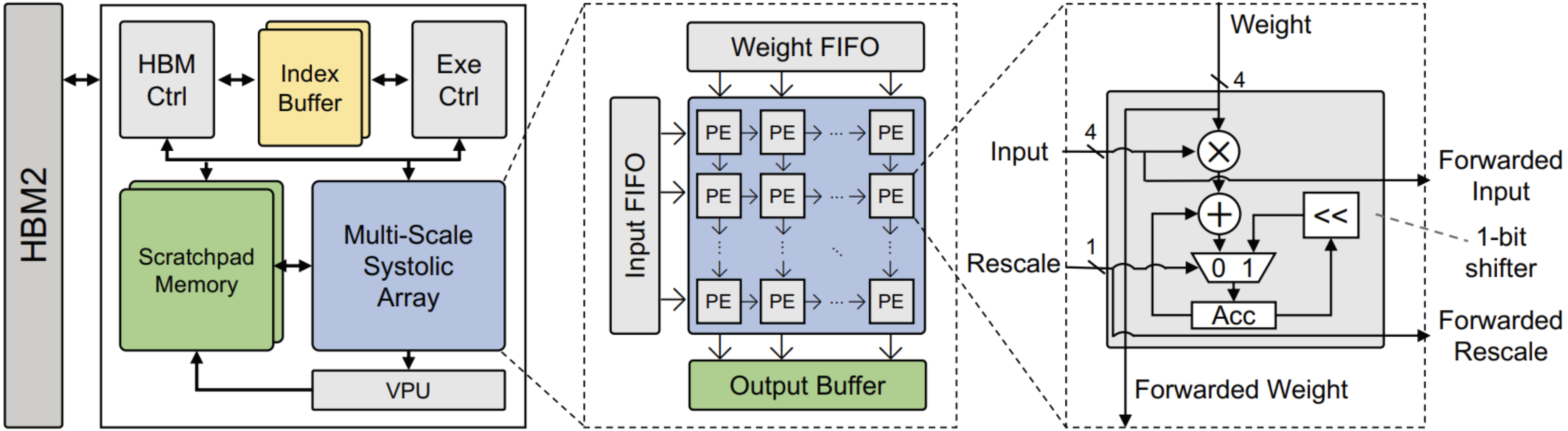


(a) (b) (c)

- **HBM Controller**: manage data movement between the on-chip buffer and **HBM2**
- **Execution Controller**: send address to the **Scratchpad Memory** to bring data, send data and control signals to the **systolic array**

Hardware Architecture

❑ 2. Multi-Scale Systolic Array (MSA)

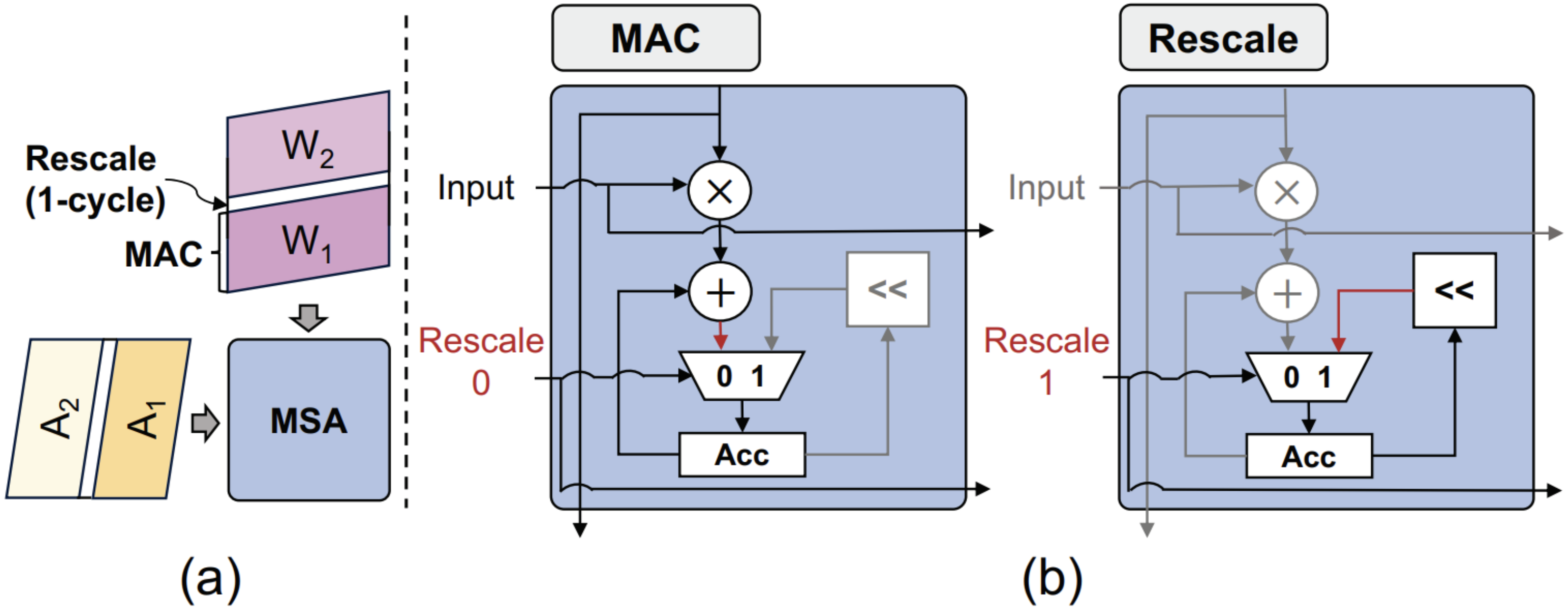


(a) (b) (c)

- 2D mesh of PEs with FIFOs attached for skewing inputs and weights
- Single 64 x 64 systolic array with each PE executing a 4-bit MAC operation
- When precision is INT8, 4 PEs are grouped to perform 8-bit multiplication

Hardware Architecture

❑ 2. Multi-Scale Systolic Array (MSA)



- Output stationary (partial sum is accumulated in each PE)
- Normal MAC operations for matrix reduction, **1-cycle bubble** for rescale
- Execution Controller has the metadata of number of channels, indices of tensor splitting points, and rescale factors

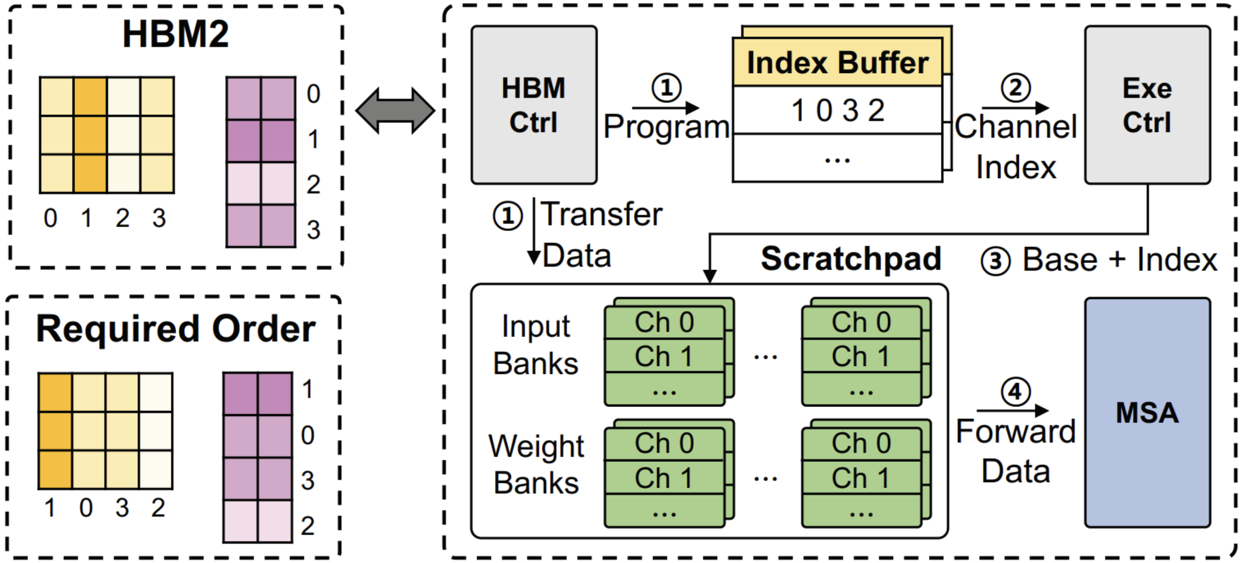
Hardware Architecture

- 3. Vector Processing Unit (VPU)
  - SIMD style FPU that operates on vector elements
  - Scaling of INT32 results from Output Buffer into INT4/INT8 before storing back to Scratchpad Memory (with an optional activation)
  - Use calibrated bias and scale factors
  - Consists of 64 FPUs and internal vector registers for pipelining
  - Additional registers to buffer scaling factors for quantization
  - Performs softmax and LayerNorms in the Transformer block

Hardware Architecture

❑ 4. Controllers & Index Buffer

- Indirect indexing



① Program: Store the computation order in the Index Buffer, which is pre-determined
② Channel Index: Execution Controller looks up in the Index buffer and obtains channel indices
③ Base + Index: Generate an address for the target channel to load
④ Forward Data: Channels are sent to the MSA in the required order
① Transfer Data: HBM Controller sends data from off-chip memory to Scratchpad Memory

Hardware Architecture

❑ 5. Scratchpad Memory & Output Buffer

- Scratchpad Memory: All inputs and weights are quantized into INT4/INT8 and stored in it

- Output Buffer: Stores the computation result from the MSA in INT32 and sends them to the VPU for rescaling back to INT4/INT8

Evaluation

- Experimental Methodology
  1. Software Implementation
     - Model: OPT, LLaMa, LlaMa-2, BERT-Large (for encoder-only model)
     - Datasets: WikiText2, Penn Treebank(PTB)
     - Evaluation: Perplexity
  2. Quantization Baselines
     - Compare with SmoothQuant, ANT, OliVe
  3. Hardware Implementation
     - Implement & Verification: SystemVerilog
     - DRAM performance: Ramulator
  4. Accelerator Baselines
     - Compare with OLAccel, ANT, OliVe

Evaluation

❑ Language Model Performance

- PTQ Performance on LLMs

- Disable quantization of Tender for matrix multiplication between activations
- Sequence length = 2048

INT8/INT4 PTQ results (perplexity) for LLMs

Precision	Scheme	OPT-6.7B		OPT-13B		OPT-66B		Llama-2-7B		Llama-2-13B		Llama-2-70B		LLaMA-7B		LLaMA-13B	
		Wiki	PTB	Wiki	PTB	Wiki	PTB	Wiki	PTB	Wiki	PTB	Wiki	PTB	Wiki	PTB	Wiki	PTB
FP16	Base	10.86	13.09	10.13	12.34	9.34	11.36	5.47	20.83	4.88	28.93	3.32	14.44	5.68	8.80	5.09	8.07
INT8	SmoothQuant	**10.93**	13.21	10.40	12.53	9.87	11.71	48.54	1E+4	447.52	491.51	17.30	46.96	27.85	54.98	16.02	32.84
	ANT	19.72	27.96	4E+3	3E+3	3E+3	3E+3	8.79	4E+4	20.52	152.01	7.28	36.18	8.52	13.41	7.49	10.85
	OliVe	**10.93**	13.23	10.28	12.41	**9.43**	11.41	8.16	30.12	30.50	26.16	50.94	245.09	53.34	113.48	7.62	10.76
	Tender	**10.93**	**13.14**	**10.17**	**12.39**	**9.43**	**11.40**	**5.77**	**18.95**	**5.09**	**21.13**	**3.48**	**14.23**	**5.87**	**9.05**	**5.28**	**8.27**
INT4	SmoothQuant	5E+4	2E+4	9E+3	1E+4	6E+4	3E+4	3E+5	3E+5	4E+4	4E+4	7E+4	5E+4	3E+5	2E+5	2E+5	2E+5
	ANT	9E+3	6E+3	4E+4	3E+4	1E+4	7E+3	189.72	2E+4	165.19	1E+3	24.96	155.92	80.13	109.21	96.71	247.65
	OliVe	50.83	43.96	35.76	75.37	6E+3	4E+3	44.24	860.93	1E+3	**97.93**	99.91	216.53	195.15	359.43	94.32	181.69
	Tender	**13.56**	**16.28**	**16.43**	**19.92**	**12.38**	**14.01**	**36.47**	**114.44**	**55.08**	208.76	**13.43**	**50.66**	**23.85**	**38.09**	**13.68**	**28.24**

- Almost the same perplexity as FP16 baseline
- Far better perplexity than others → Tender can well separate the outlier channels

HYPER ACCEL

Evaluation

❑ Language Model Performance

- Sequence Length Sensitivity

"Tender": Disable quantization of Tender for matrix multiplication between activations
"Tender(all)": Quantizes all the matrix multiplications

INT8/INT4 PTQ results (perplexity) across different sequence length

Precision	Scheme	2048		256		32	
		Wiki	PTB	Wiki	PTB	Wiki	PTB
FP16	Base	10.86	13.09	19.18	22.00	78.97	103.42
INT8	SmoothQuant	**10.93**	13.21	**19.17**	22.14	79.32	**102.68**
	ANT	19.72	27.96	48.43	57.97	396.01	364.00
	OliVe	**10.93**	13.23	19.24	22.29	79.69	104.42
	Tender (all)	10.98	13.19	19.31	22.08	78.93	102.99
	Tender	**10.93**	**13.14**	19.28	**22.06**	**78.81**	102.84
INT4	SmoothQuant	5E+4	2E+4	5E+4	2E+4	4E+4	2E+4
	ANT	9E+3	6E+3	8E+3	6E+3	6E+3	3E+3
	OliVe	50.83	43.96	88.05	113.53	441.03	371.73
	Tender (all)	17.15	23.25	27.57	30.58	96.34	118.85
	Tender	**13.56**	**16.28**	**23.16**	**26.12**	**91.27**	**111.90**

- Tender shows the best performance for most of the quantization scenarios & maintains the perplexity close to FP16 baseline

- Even Tender(all) outperforms the prior works that do not quantize matrix multiplication between activations

HYPER ACCEL

Evaluation

❑ Language Model Performance

- Quantization Accuracy on BERT (with GLUE benchmark)

INT8/INT4 PTQ results (accuracy) on BERT-Large

Precision	Scheme	CoLA	SST-2	MRPC	STS-B	QQP	QNLI
FP32	Base	60.20	93.12	91.58	89.94	91.40	92.33
INT8	ANT	59.16	92.55	77.99	89.23	89.66	81.48
	OliVe	**61.12**	93.12	91.33	89.91	91.42	92.02
	Tender	60.45	**93.23**	**91.55**	**89.98**	**91.43**	**92.31**
INT4	ANT	53.77	90.60	21.09	85.93	83.62	60.86
	OliVe	59.02	92.09	85.32	87.43	**89.72**	**90.48**
	Tender	**61.78**	**92.32**	**89.42**	**87.77**	89.23	90.29

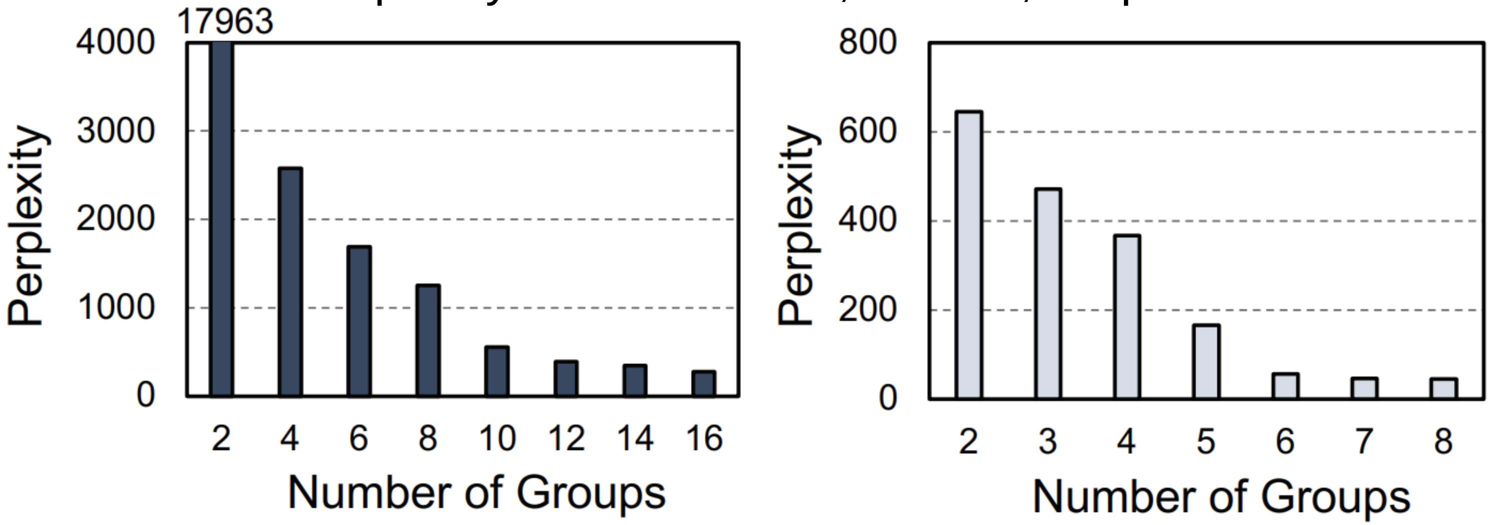
Although the outliers of the BERT-Language are much smaller than the ones of other LLMs,
Tender outperforms other baselines in many tasks
→ Tender also benefits encoder-only and relatively small models

Evaluation

❑ Language Model Performance

- Multi-Scale Quantization

Perplexity on Llama-2-7B, w/ PTB, Sequence = 256



(a) INT4 (b) INT8

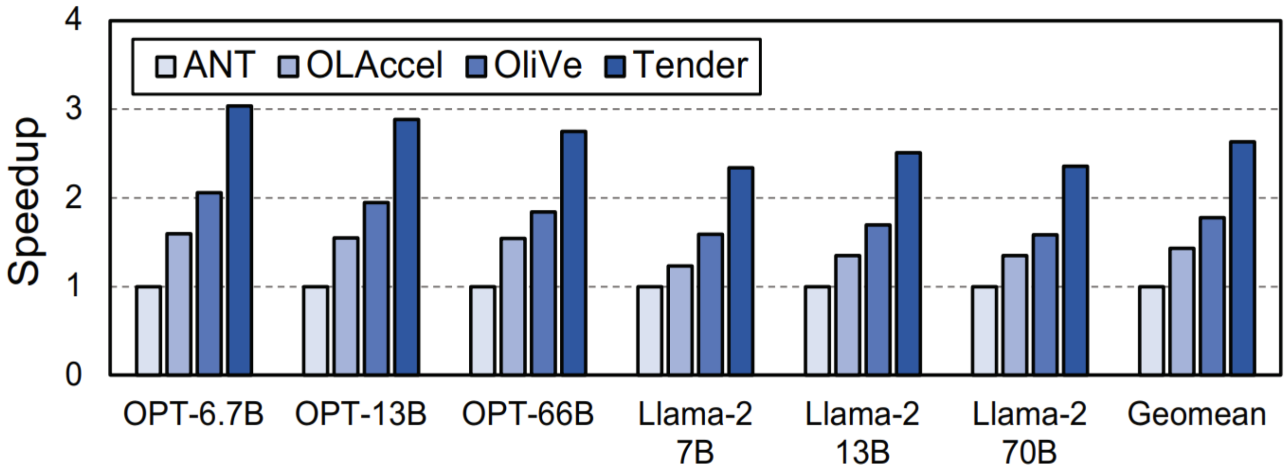
→ Decomposing the channels into multiple groups is necessary to achieve better performance

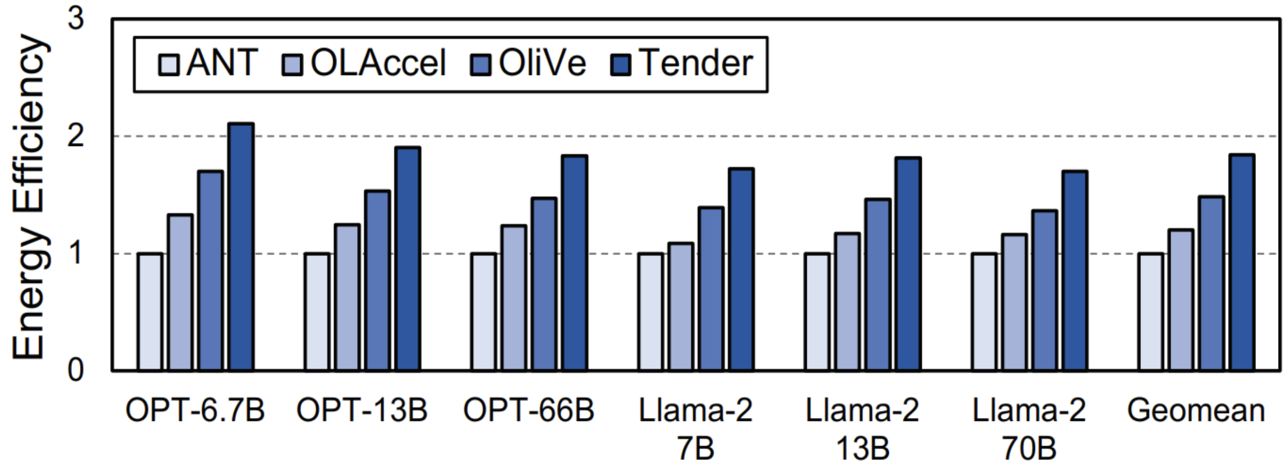
Evaluation

❑ Tender Performance

$2.63\times, 1.84\times, 1.48\times$ speedups &
$1.84\times, 1.53\times, 1.24\times$ energy efficiency
over ANT, OLAccel, OliVe

Performance & Energy Efficiency (INT4 Precision)





HYPER ACCEL

Conclusion

- Tender minimizes quantization errors by splitting the activation tensors along the feature/channel dimension to separate the outlier channels

- Tender addresses the runtime overhead of the channel decomposition by implicit requantization with a minimally extended systolic array

- Tender significantly improves PTQ performance even for ultra low-bit quantization without mixed-precision or custom datatypes