Jetfire : Efficient and Accurate Transformer Pretrining with INT8 Data Flow and Per-Block Quantization

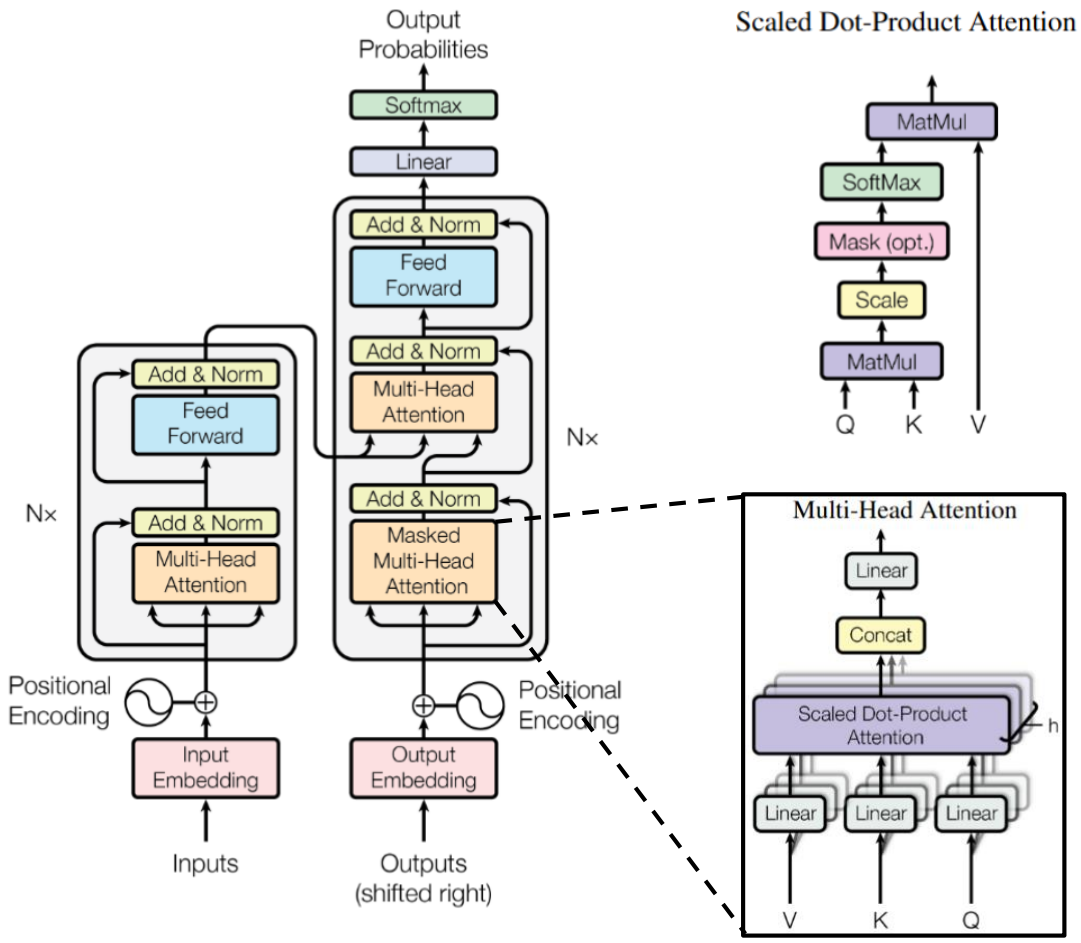
Haocheng Xi, Yuxiang Chen, Kang Zhao, KAI JUN TEH, Jianfei Chen, Jun Zhu

Presenter : Taehyeon Kim, Yongha Shin, Sanghyeon Cho

Contents

1 Introduction

- Large-scale pre-trained transformer-based models e.g. GPT-4, LLAMA, PaLM

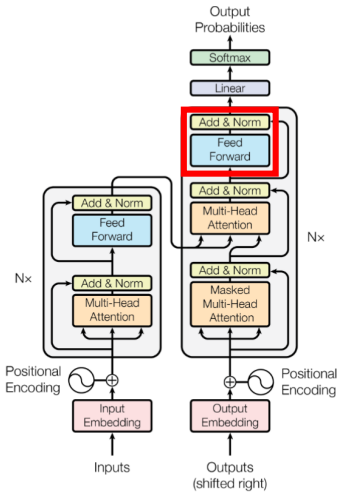


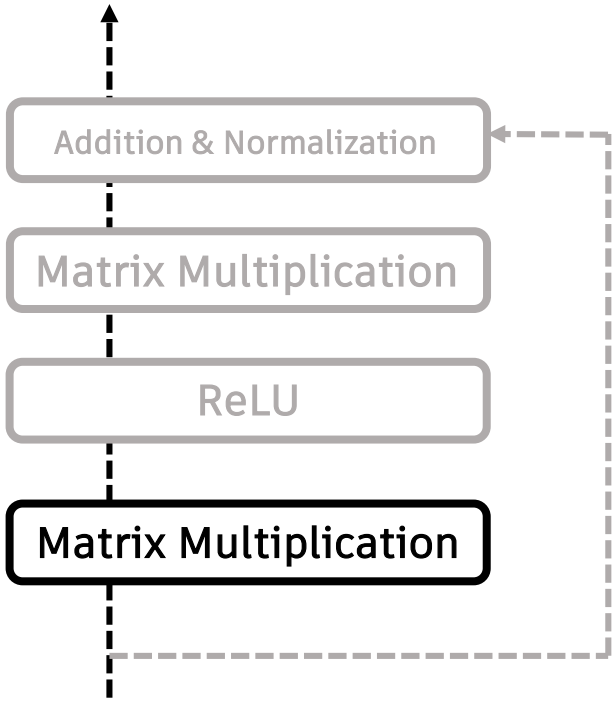
A. Vaswani, “Attention is all you need”

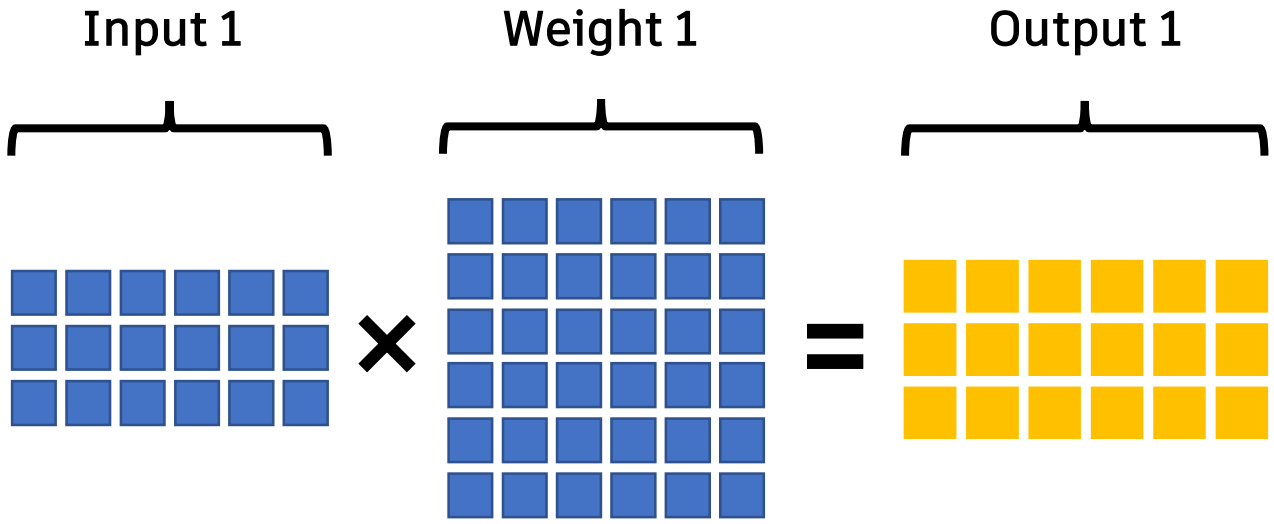
1 Introduction

- Pre-training transformers require numerous **computations** & high-bandwidth **memory**

e.g. sub-layer of FP16 FFN





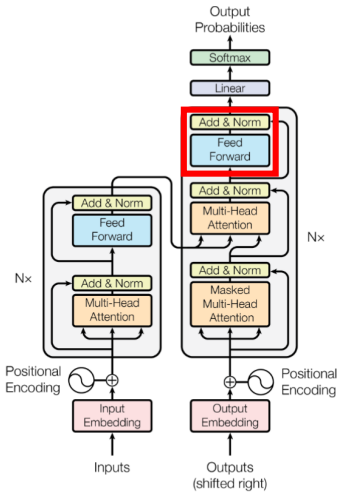


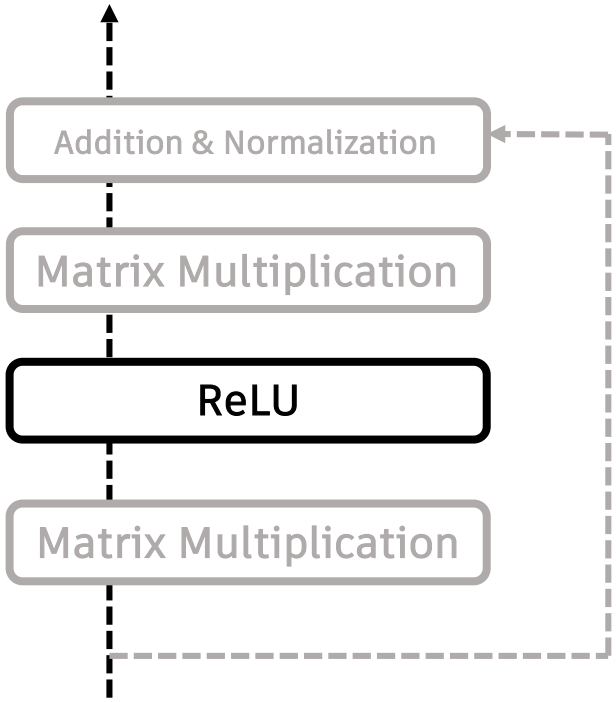
Computations : 198 FLOPs
Memory access : 72 bytes

1 Introduction

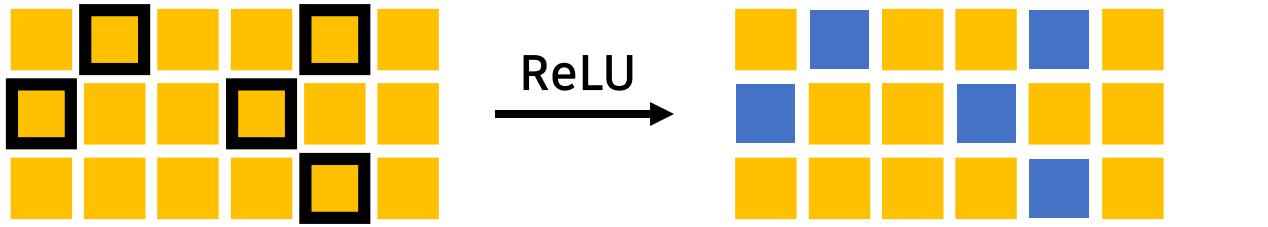
- Pre-training transformers require numerous **computations** & high-bandwidth **memory**

e.g. sub-layer of FP16 FFN





$$\mathbf{ReLU}(\mathbf{x}) = \mathbf{max}(\mathbf{0}, \mathbf{x})$$

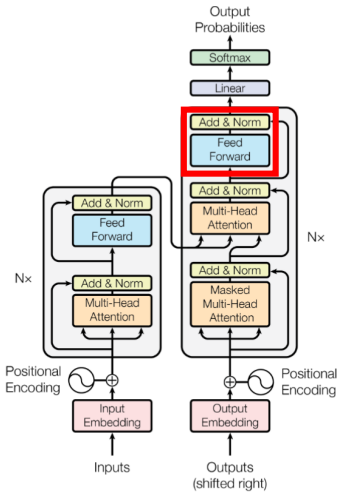


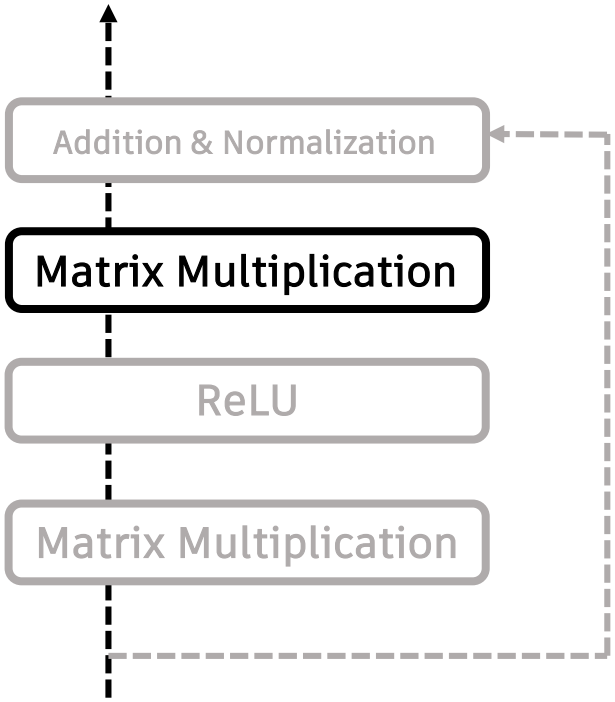
Computations : 198 + 18 = 216 FLOPs
Memory access : 72 bytes

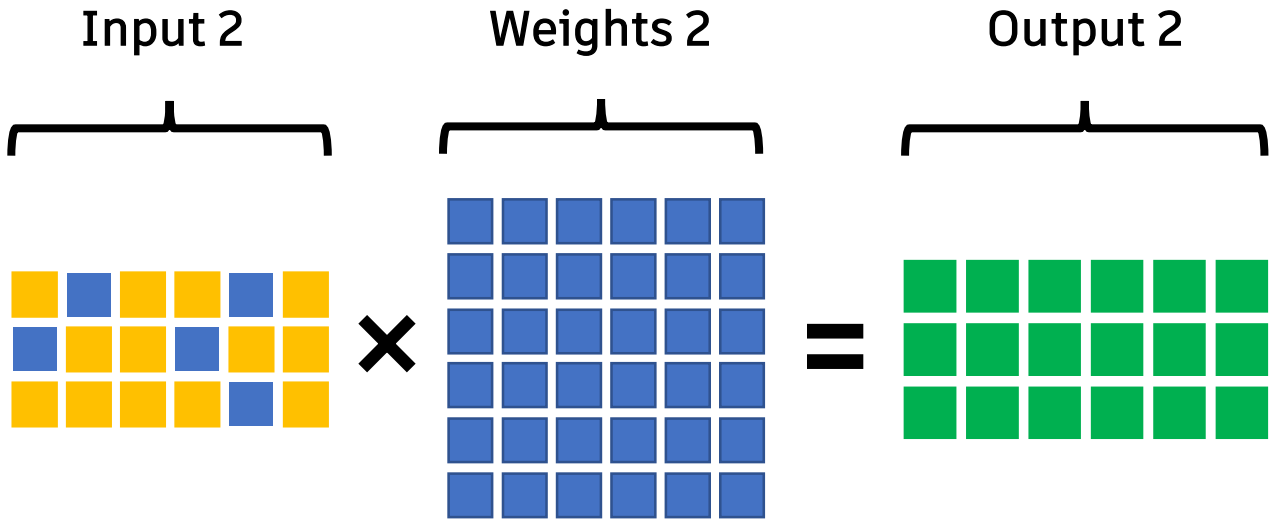
1 Introduction

- Pre-training transformers require numerous **computations** & high-bandwidth **memory**

e.g. sub-layer of FP16 FFN





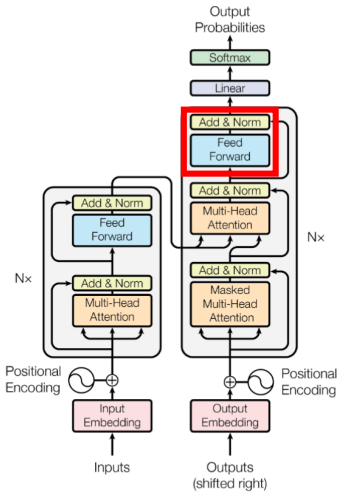


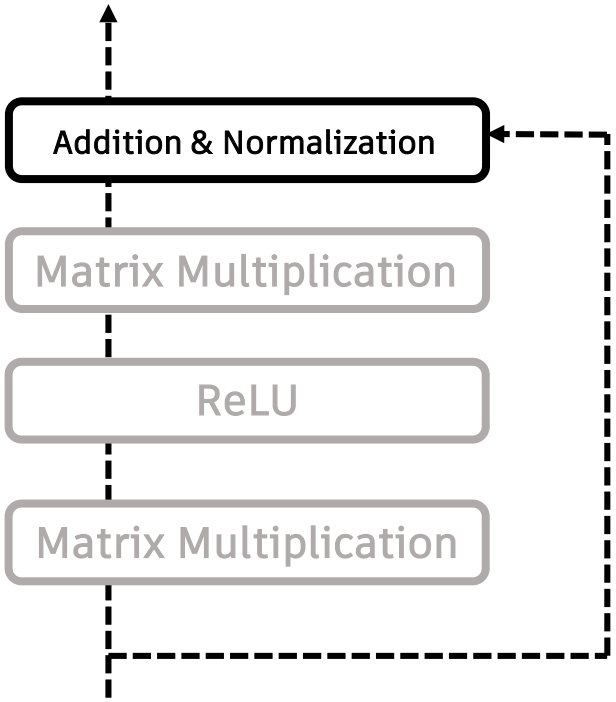
Computations : 216 + 198 = 414 FLOPs
Memory access : 72 + 72 = 144 bytes

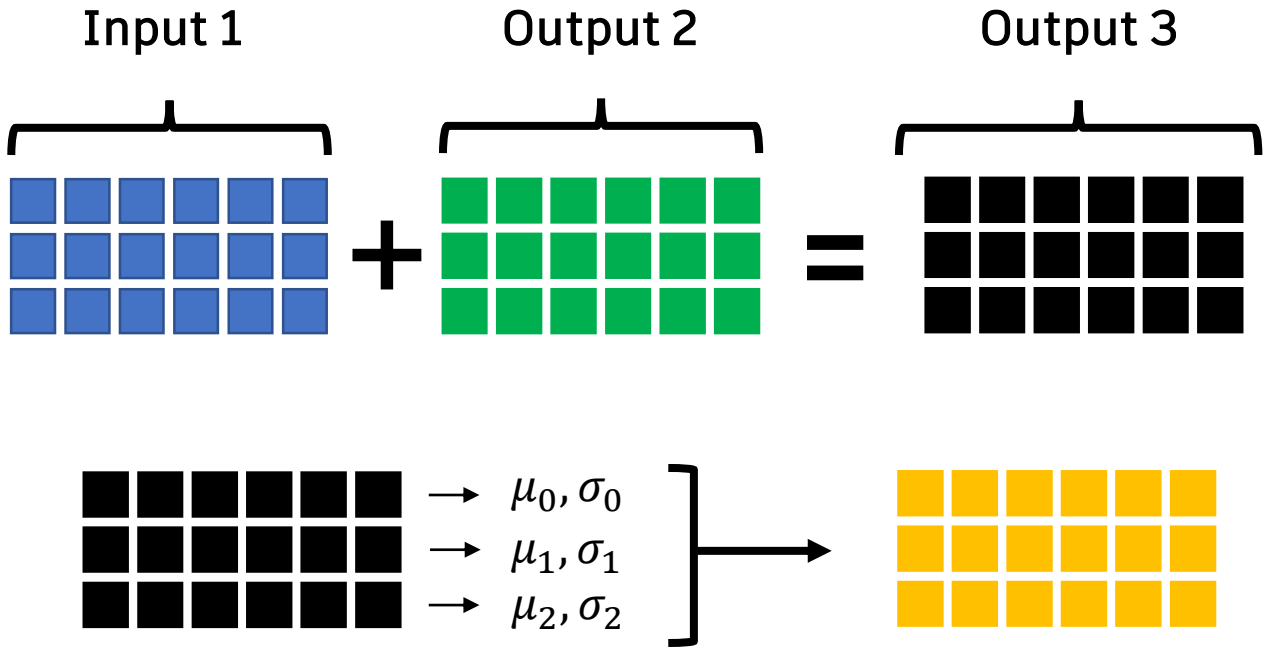
1 Introduction

- Pre-training transformers require numerous **computations** & high-bandwidth **memory**

e.g. sub-layer of FP16 FFN



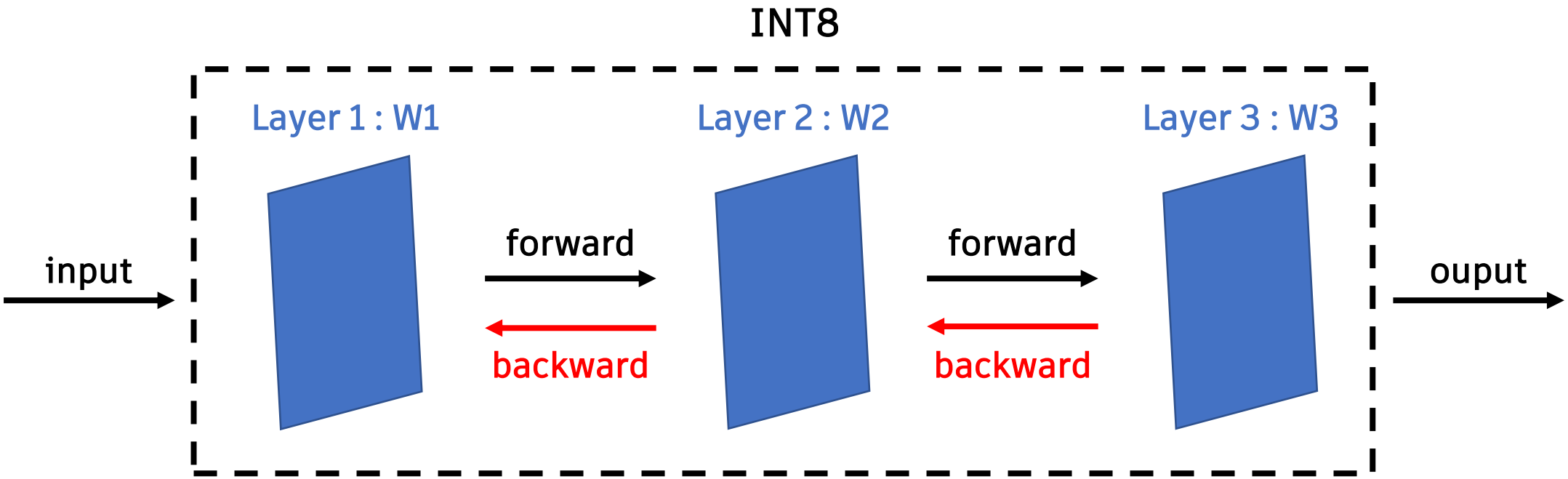




Computations : 414 + 162 = 576 FLOPs
Memory access : 144 + 36 = 180 bytes

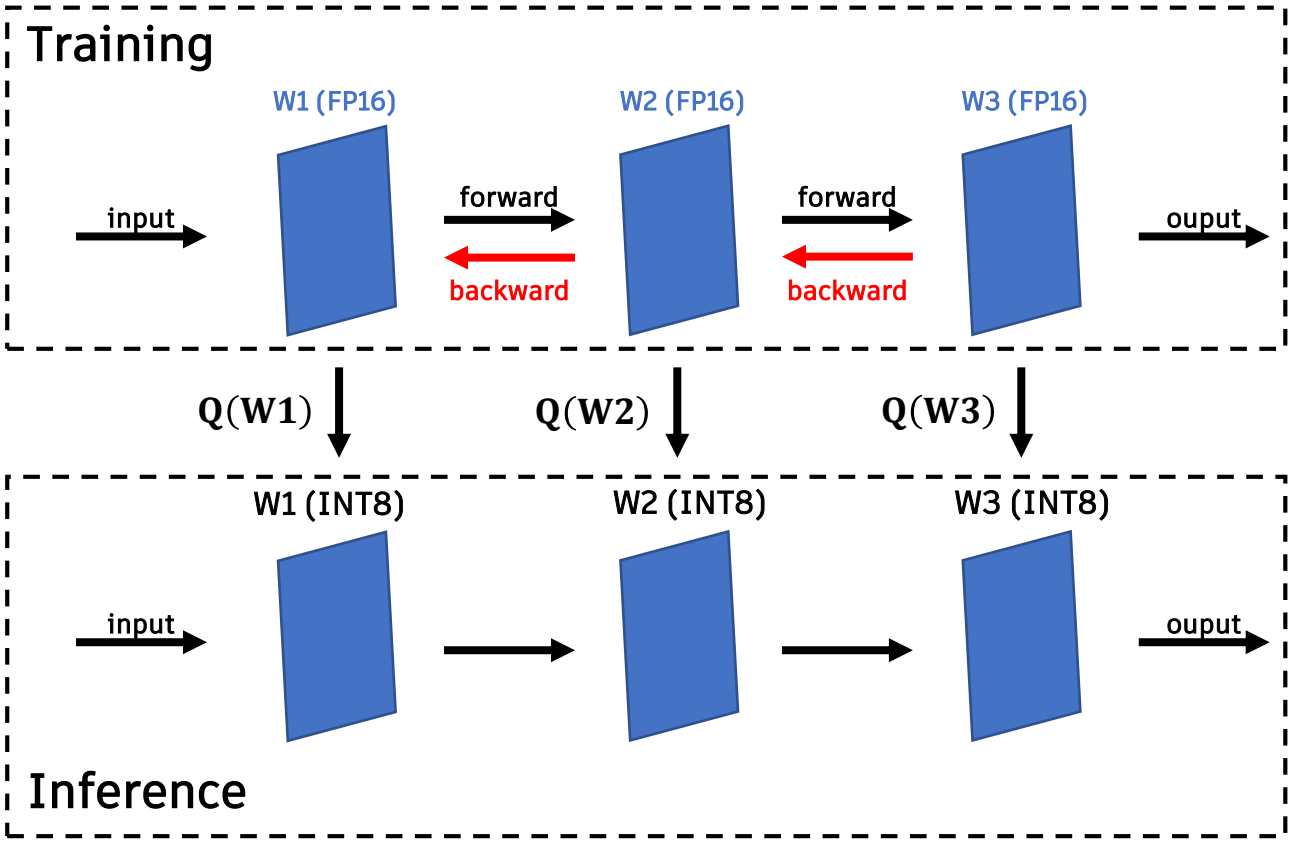
1 Introduction

- Fully Quantized Training (FQT) with INT8
  - Computation : FLOP ⇒ Integer operation
  - Memory bandwidth reduce

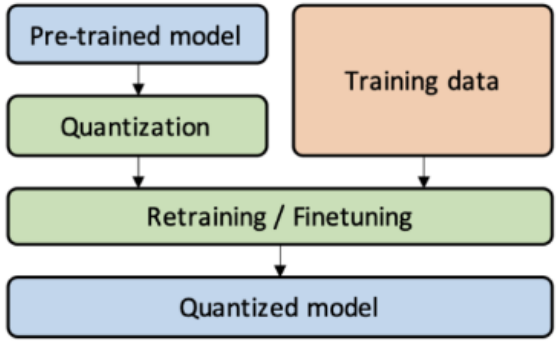


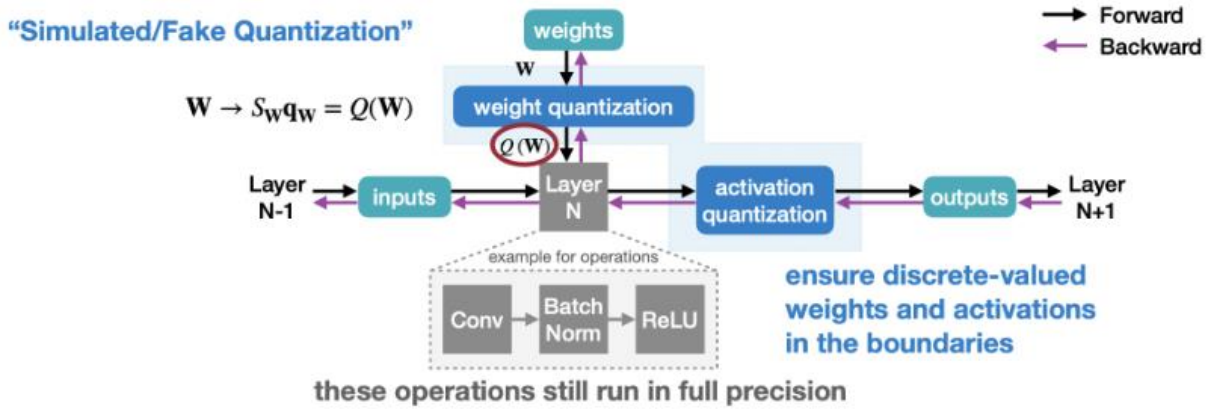
2 Related Work

- Post-Training Quantization
  - No speedup for training
  - Quantization error for output



- Quantization-Aware Training
  - focus on training accuracy, not speedup
  - Weight still full-precision



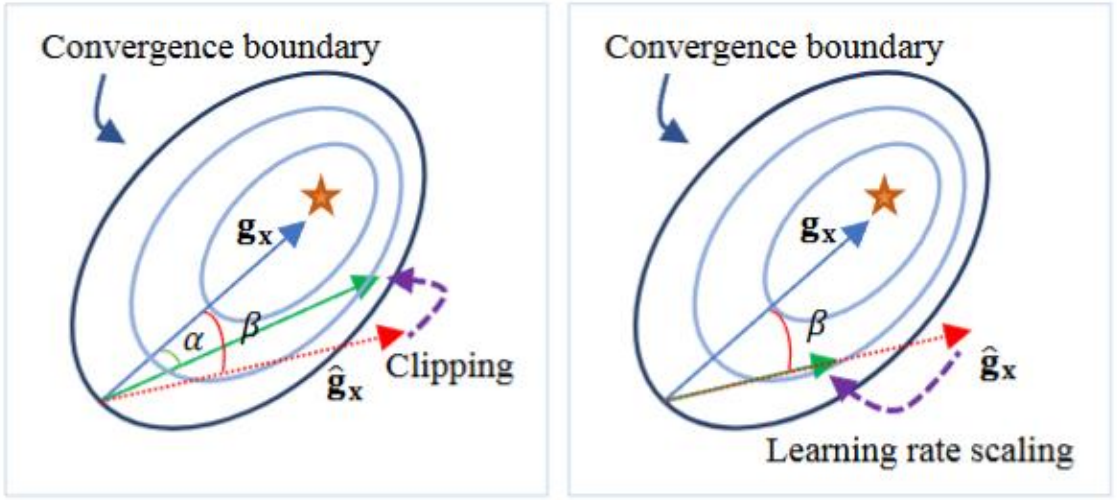


Jaeho Lee, "Linear quantization and advanced stuffs"

2 Related Work

- Fully Quantized Training : INT8 training for CNNs

- Direction Sensitive Gradient Clipping
- Deviation Counteractive Learning Rate Scaling



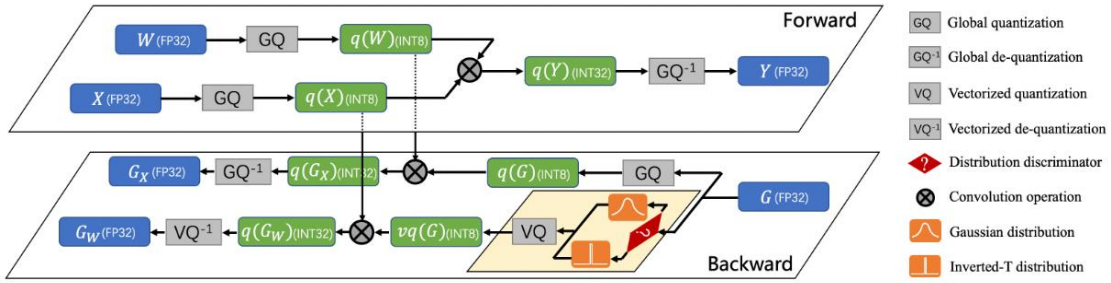
F. Zhu, arxiv:1912.12607v1

- Gradient Vectorized Quantization

$$\hat{G}_W = \left[\hat{G}_{W_1}, \cdots \hat{G}_{W_{C_{out}}}\right]^T, \hat{G}_{W_i} = q(G_{W_i}) \cdot \frac{S_x}{127} \cdot \frac{S_i}{127}$$

- Magnitude-aware Clipping Strategy

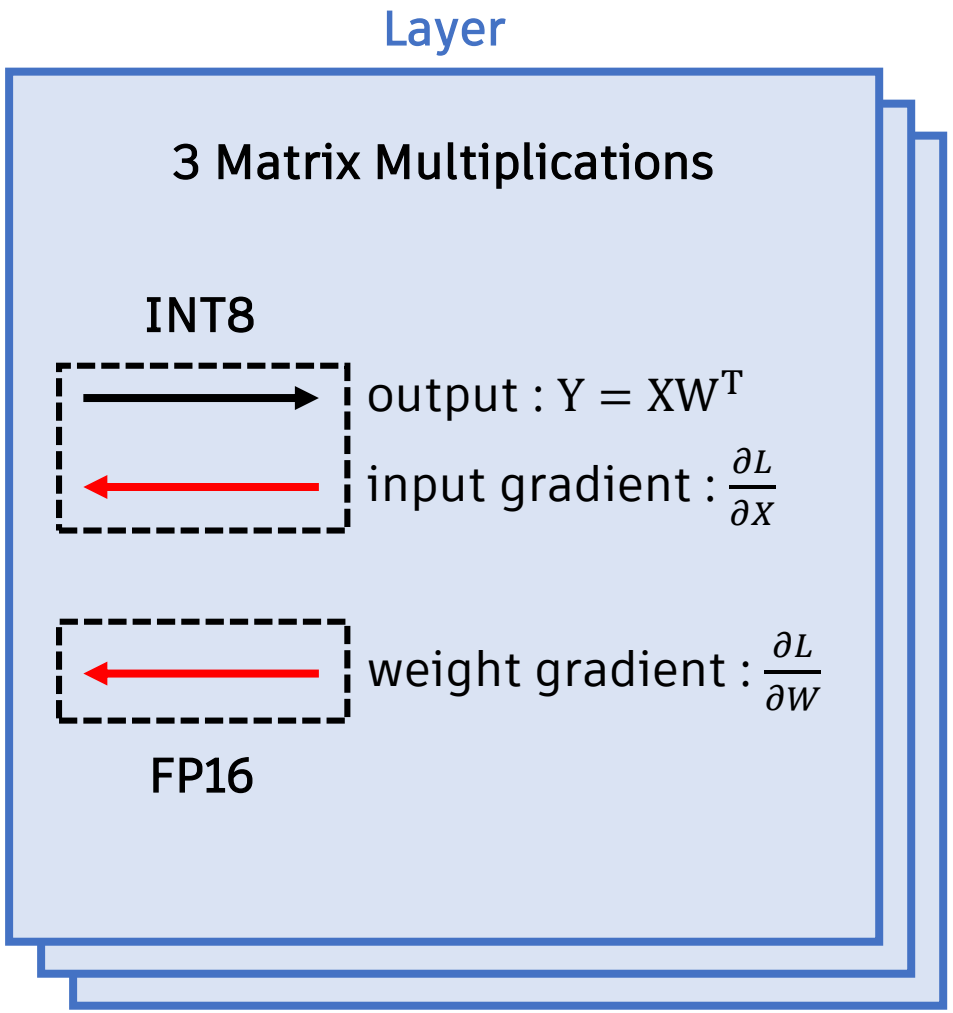
$$E = \int_{g_{min}}^{g_{max}} |g - \hat{g}| f(g) p(g) dg$$

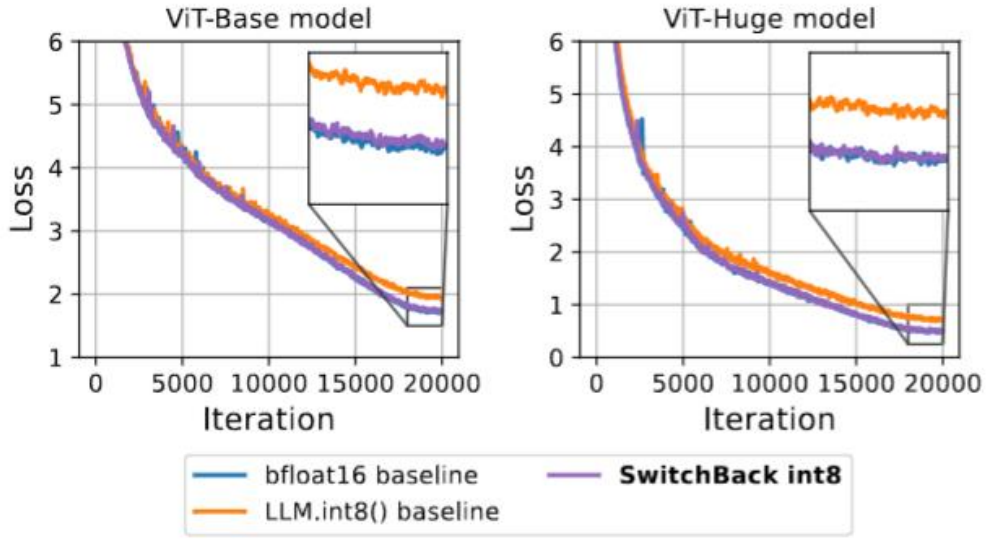


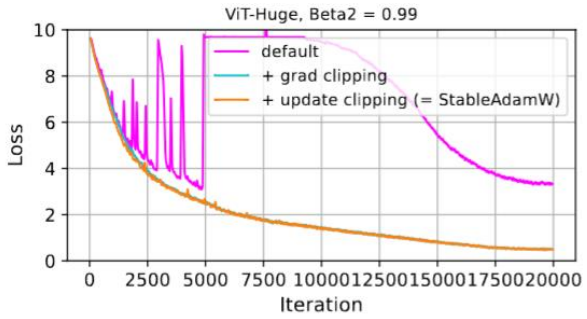
K. Zhao, “Distribution Adaptive INT8 Quantization for training CNNs”

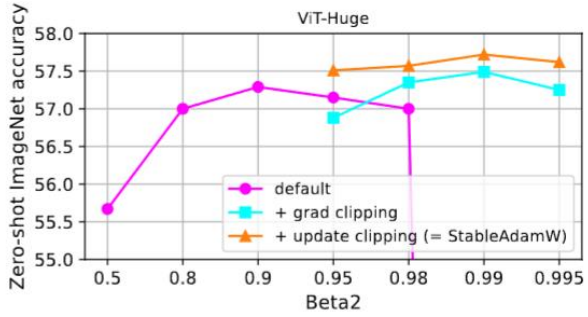
2 Related Work

- Fully Quantized Training : SwitchBack





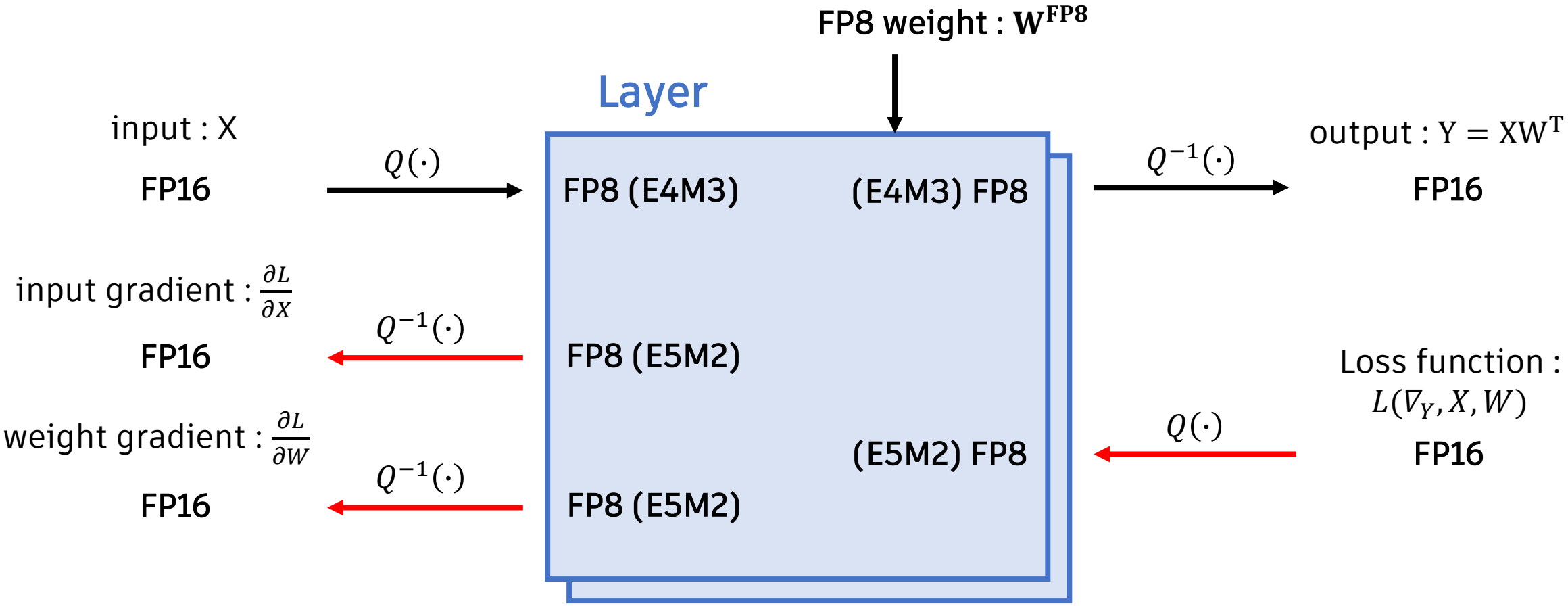




M. Wortsman, arxiv:2304.13013v2

2 Related Work

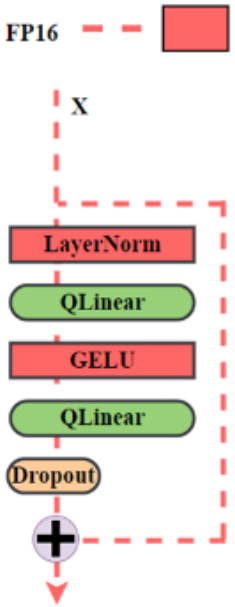
- Fully Quantized Training : FP8 training with **Hopper architecture**

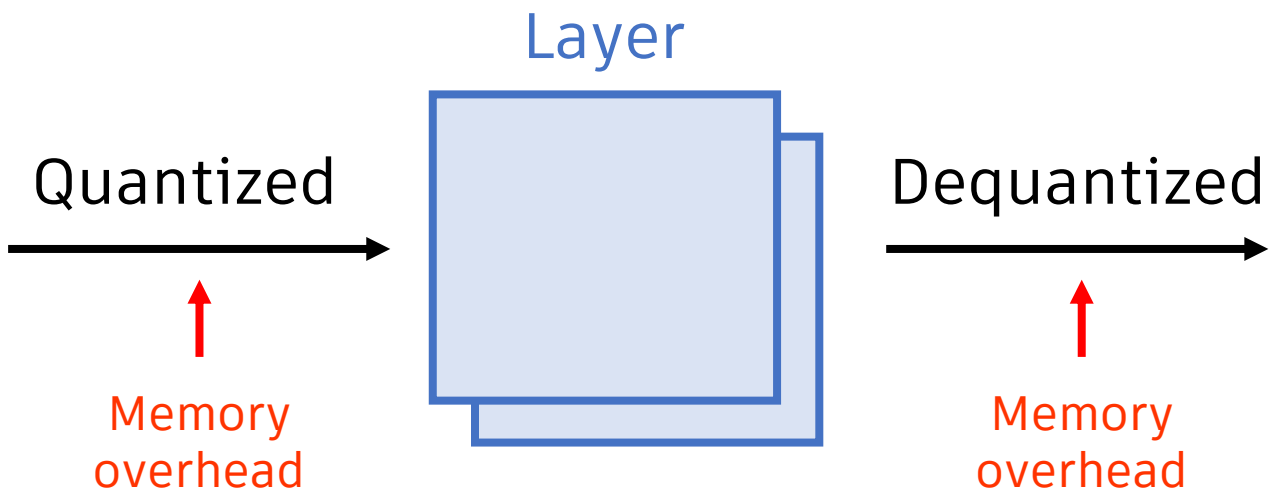


2 Related Work

- Limitations on existing FQT

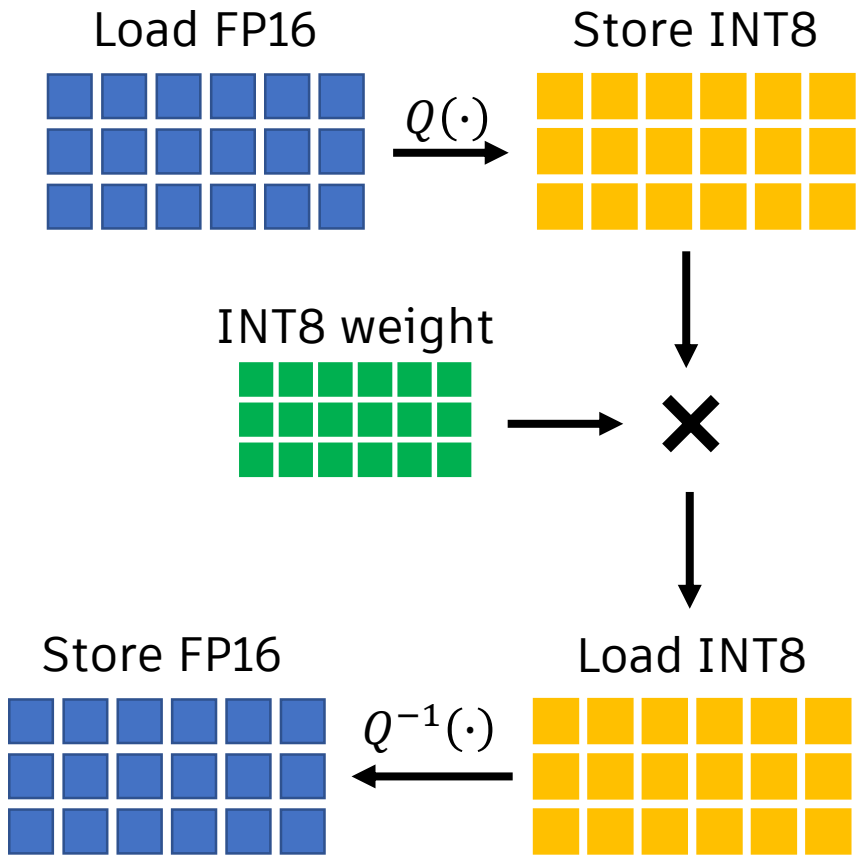
1. Previous FQT methods designed for CNN, low accuracy for Transformer
2. Most FQT focus on computation reduction, not data access overhead
3. Some FQT techniques (FP8) requires specialized hardware



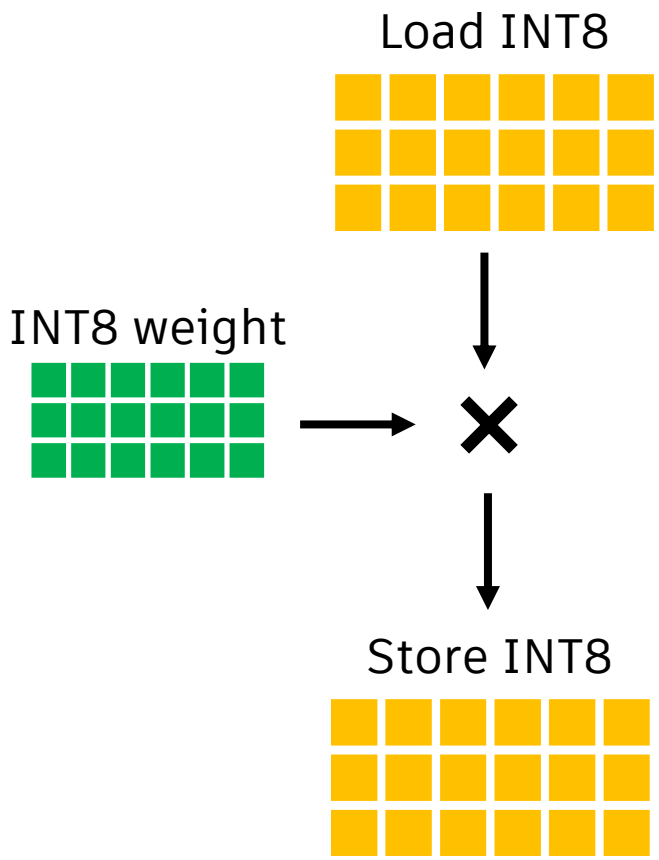


3 INT8 Data Flow

- Previous

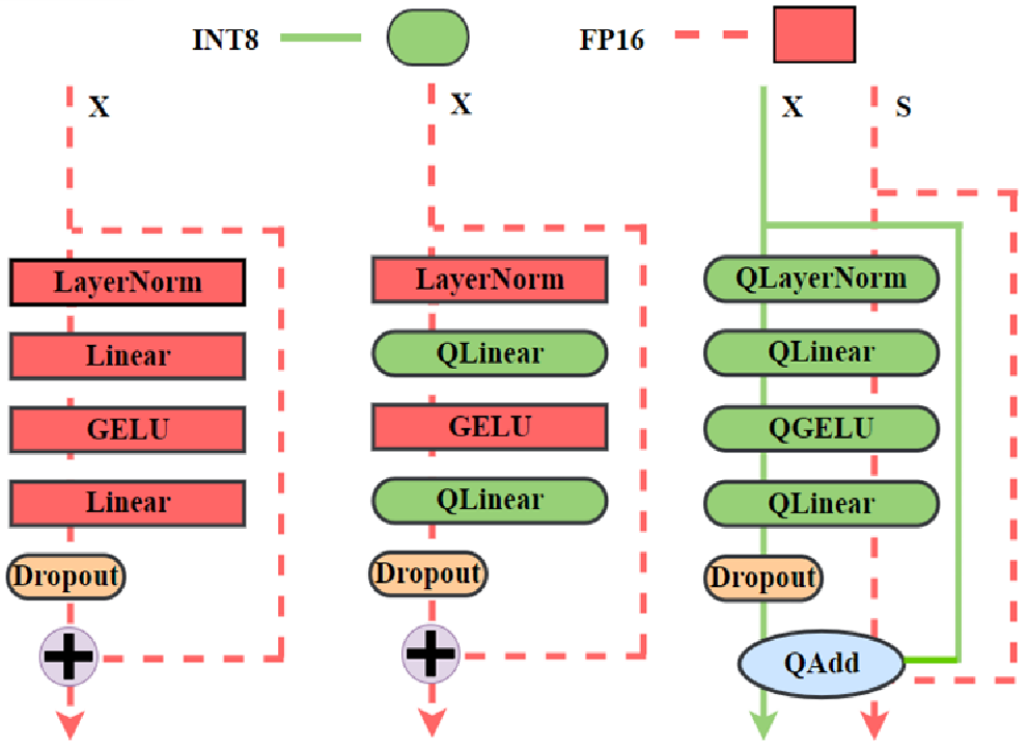


- Jetfire



3 INT8 Data Flow

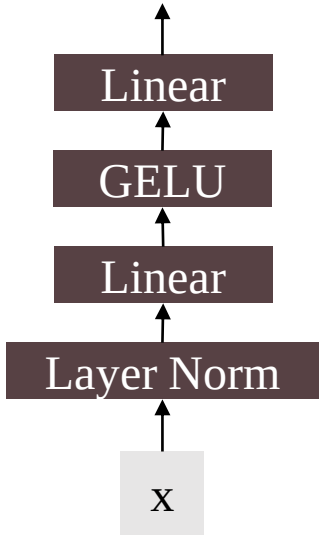
- Reduce computation and amount of memory access
- Accelerate nonlinear operators
- Activation memory consumption and amount of communication reduced



H. Xi, arxiv:2403.12422v2

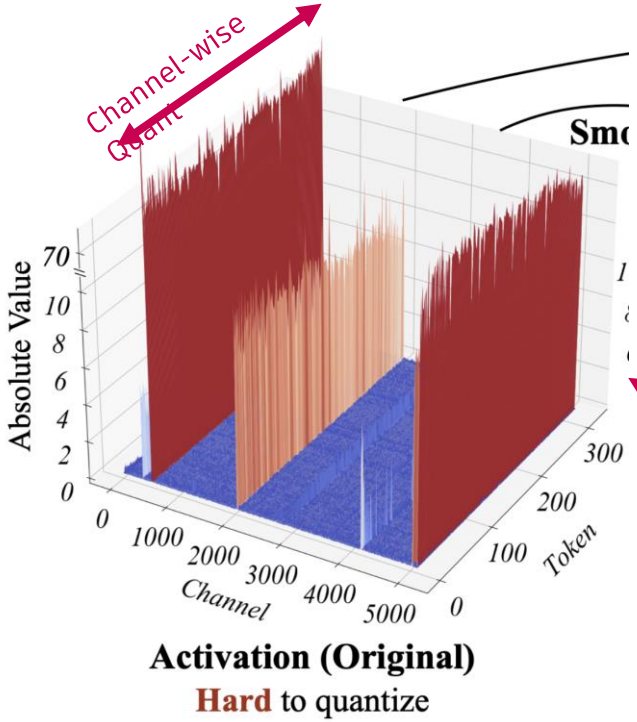
4 Method

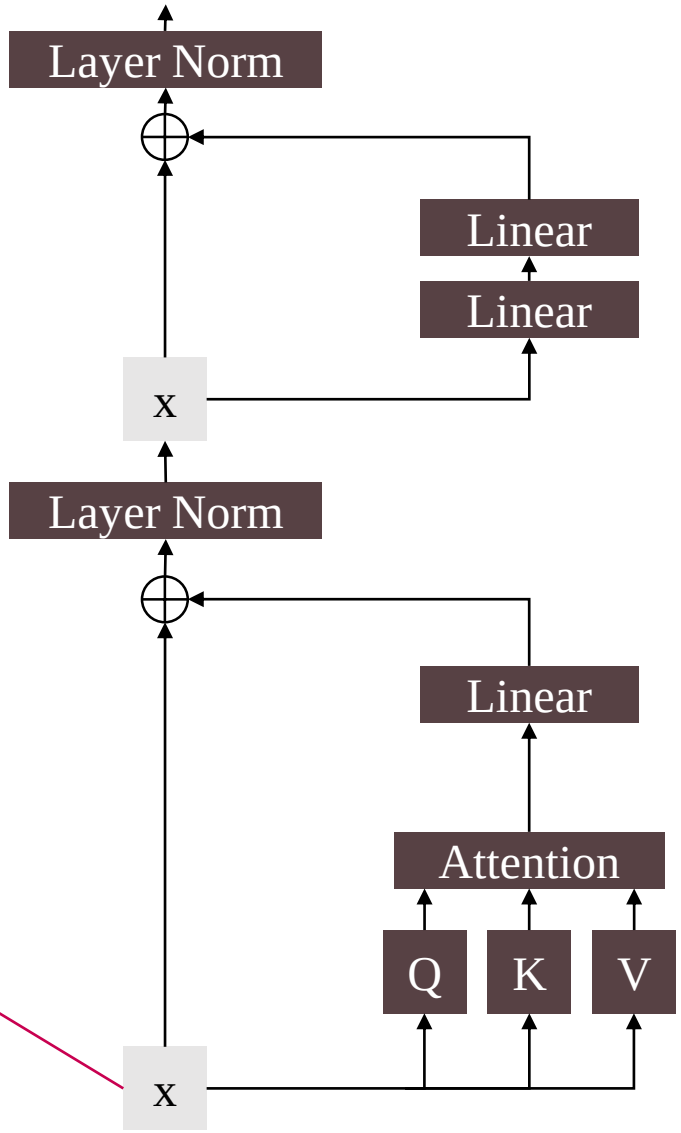
1) **Per Block Quantization**
2) Linear Layer Operator
3) Non-Linear Layer Operator



4 Method Per Block Quantization

Per Channel Quantization is advantageous for activations
due to the presence of channel-wise outliers

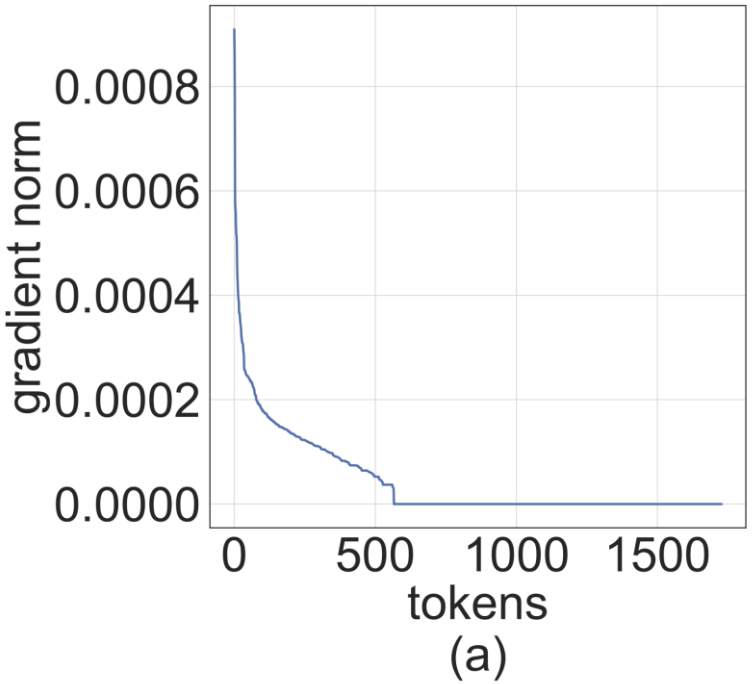




Xiao et al, (2022). SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models.

4 Method Per Block Quantization

Per Token Quantization is advantageous for gradient
due to the presence of token-wise outliers



Xi et al, (2023). Training Transformers with 4-bit Integers.

4 Method

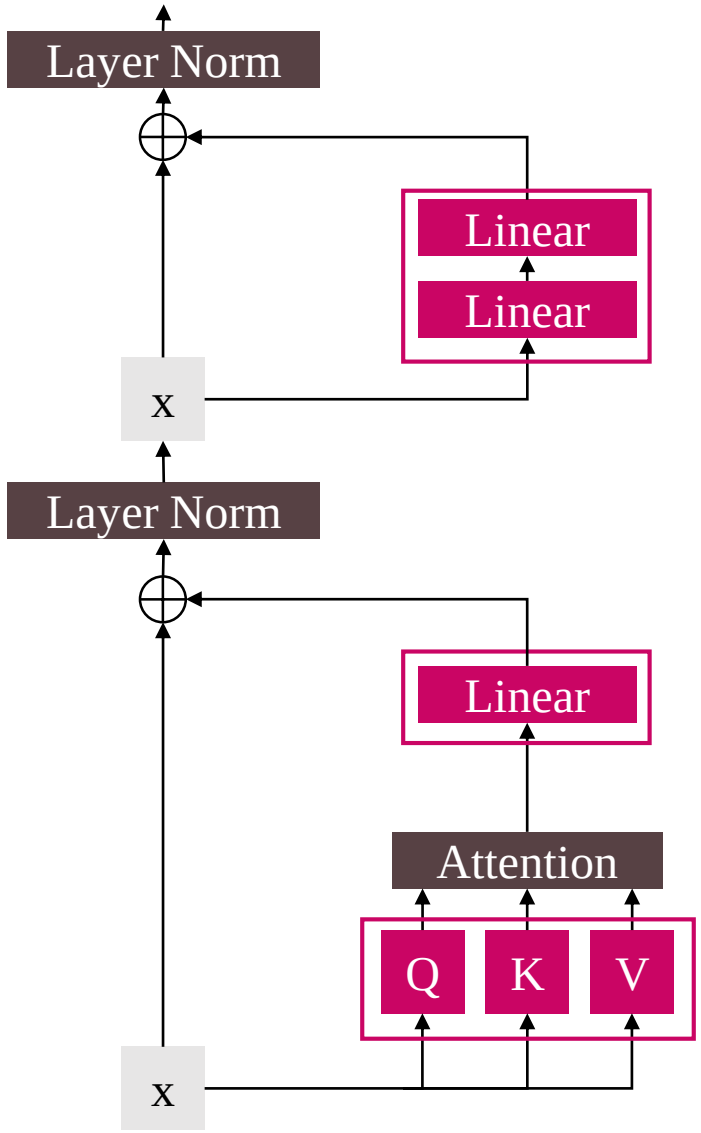
Per Block Quantization

INT8 numerical format must accurately support the following three MMs of a linear layer

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^{\mathrm{T}}$$

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

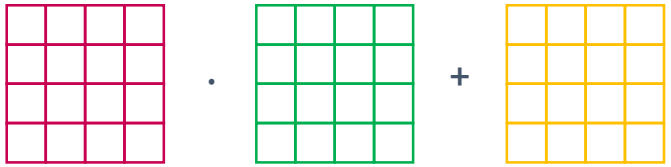
$$\nabla_{\mathbf{W}} = \nabla_{\mathbf{Y}}^{\mathrm{T}}\mathbf{X}$$



4 Method Per Block Quantization

But, using per-channel for forward pass and per-token quantization for backward pass is not hardware-friendly

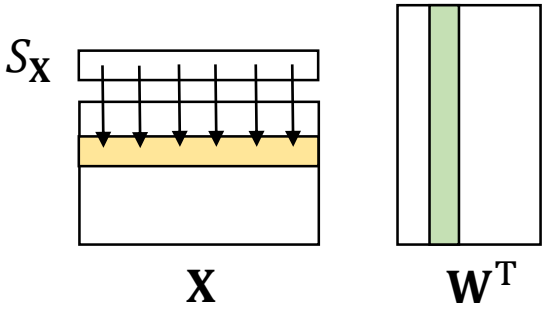
WMMA (Warp Matrix Multiply-Accumulate)



16 x 16 x 16

$\sum_K \mathbf{X}_{ik}\mathbf{W}_{kj}$, where $\mathbf{X}_{ik}, \mathbf{W}_{kj}$ are block

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^{\mathrm{T}}, \quad \nabla_{\mathbf{W}} = \nabla_{\mathbf{Y}}^{\mathrm{T}}\mathbf{X}$$

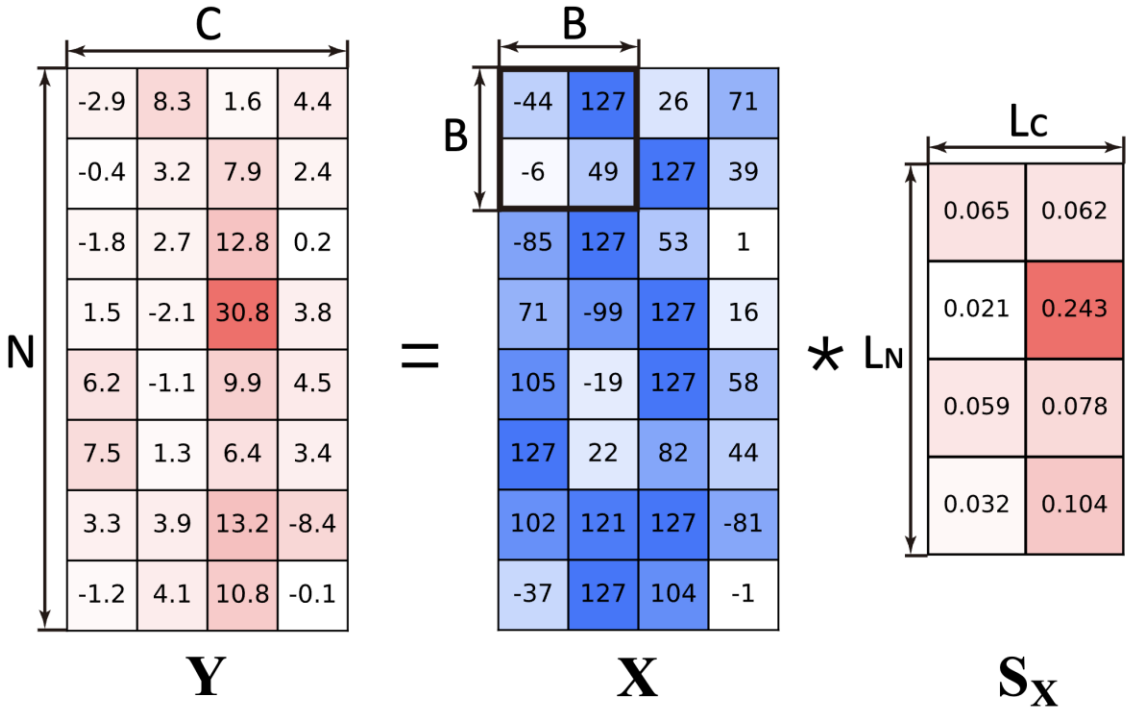


$$\sum_K S_{\mathbf{X}k}\mathbf{X}_{ik}\mathbf{W}_{kj}$$

$S_{\mathbf{X}k}$ prevents the utilization of high-speed matrix multiplication (MM) in hardware

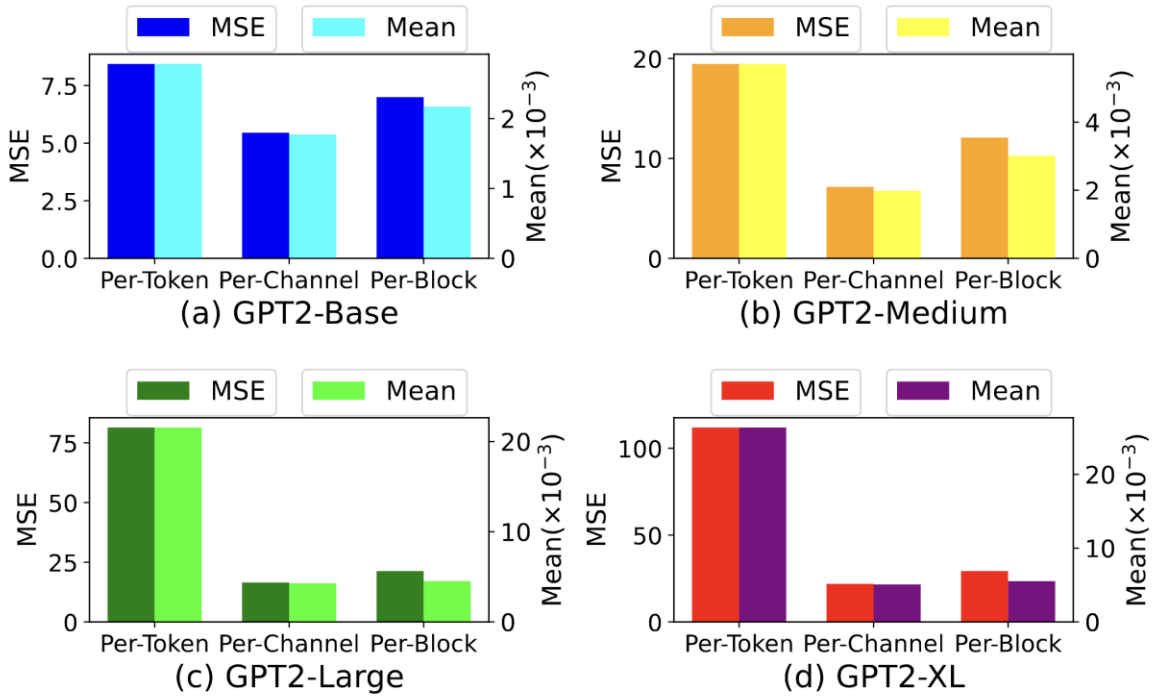
4 Method Per Block Quantization

Per-block quantization can achieve computational efficiency
and preserve accuracy at the same time



Per-block quantization

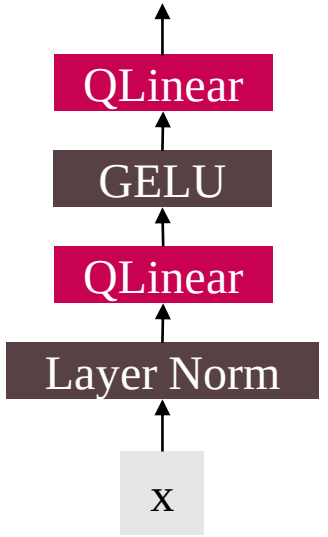
Final layer activation quant error



Per-block quantization achieved an error rate that falls
between that of per-channel and per-token quantization

4 Method

1) Per Block Quantization
2) **Linear Layer Operator**
3) Non-Linear Layer Operator



4 Method Linear Layer Operator

Jetfire uses **3-Level Tiling of Matrix Multiplication**
based on the GPU architecture

Level 1: CUDA thread block

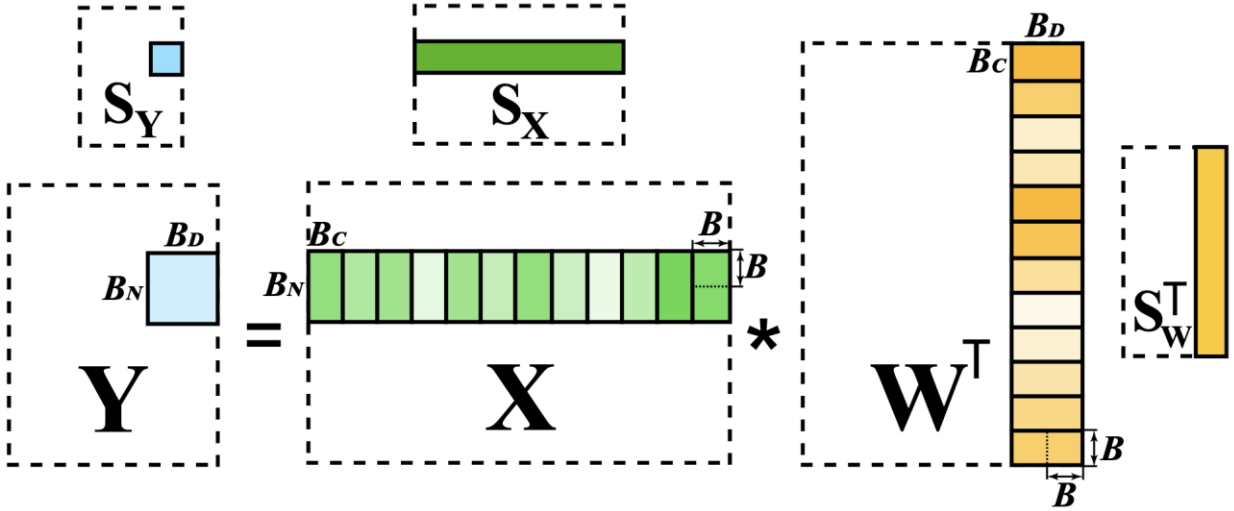
Level 2: Quantization block

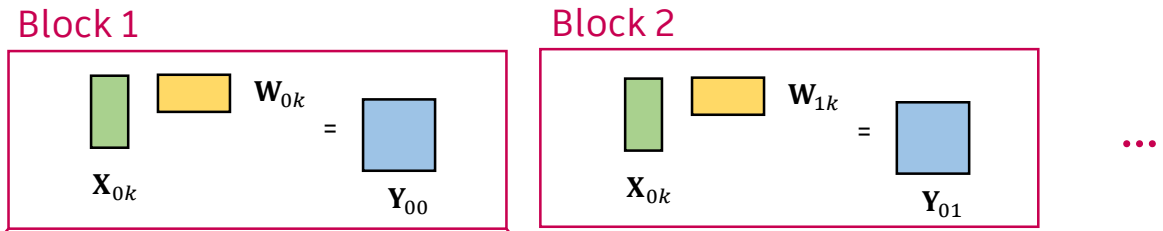
Level 3: WMMA operation

4 Method Linear Layer Operator

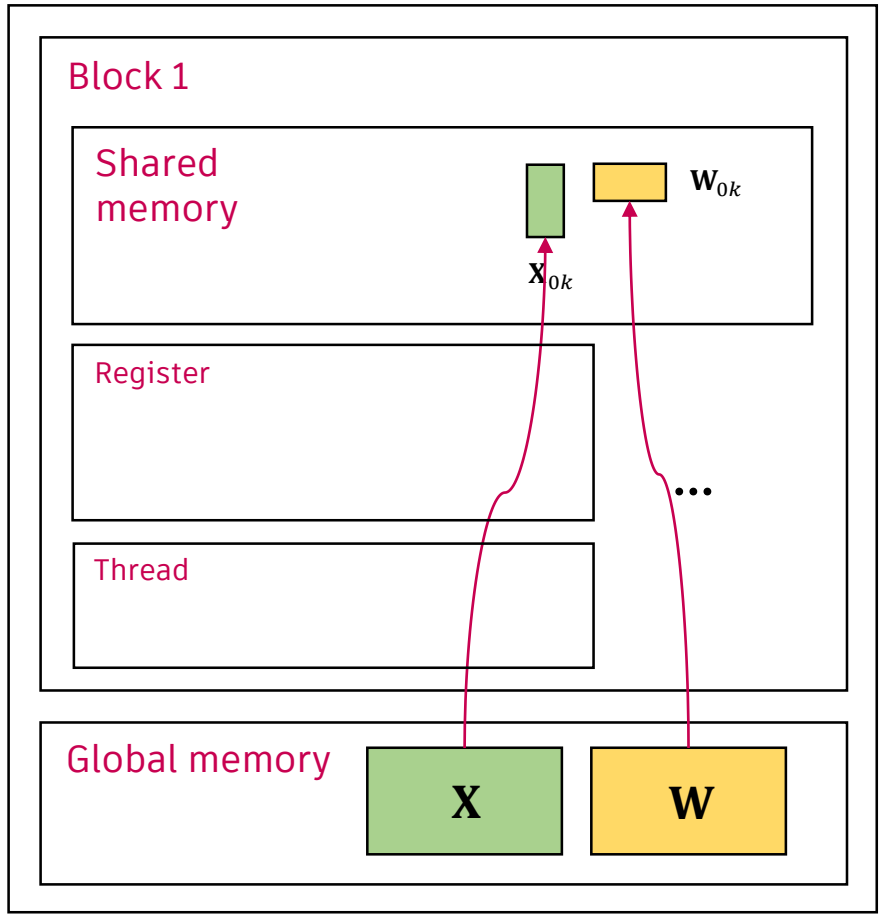
Level 1: CUDA thread block

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^{\mathrm{T}}, \text{where } \mathbf{X} \in \mathbb{R}^{N \times C}, \mathbf{W} \in \mathbb{R}^{D \times C}, \mathbf{Y} \in \mathbb{R}^{N \times D},$$





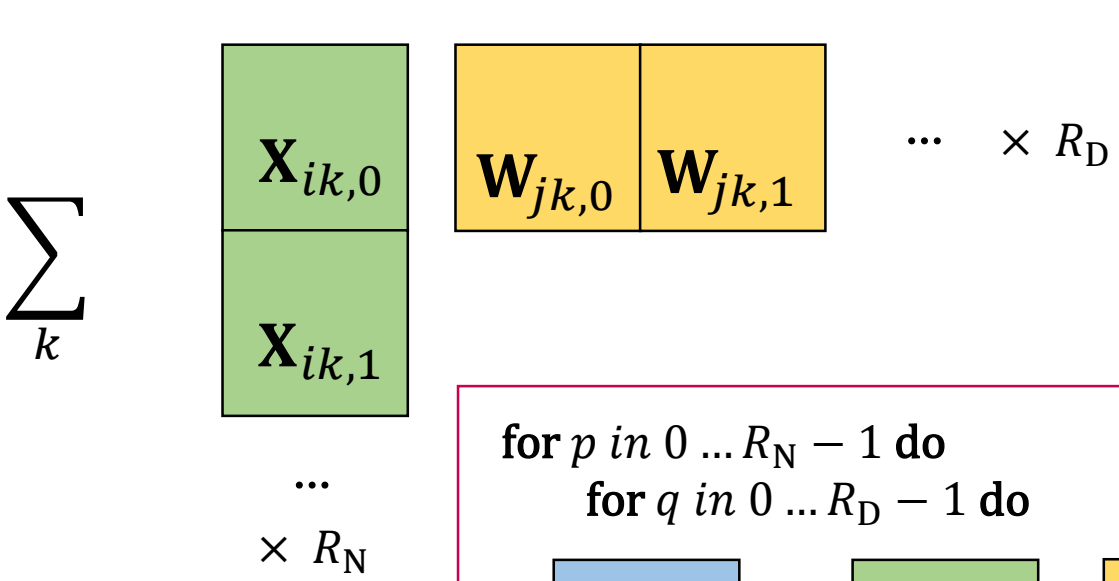
Level 2



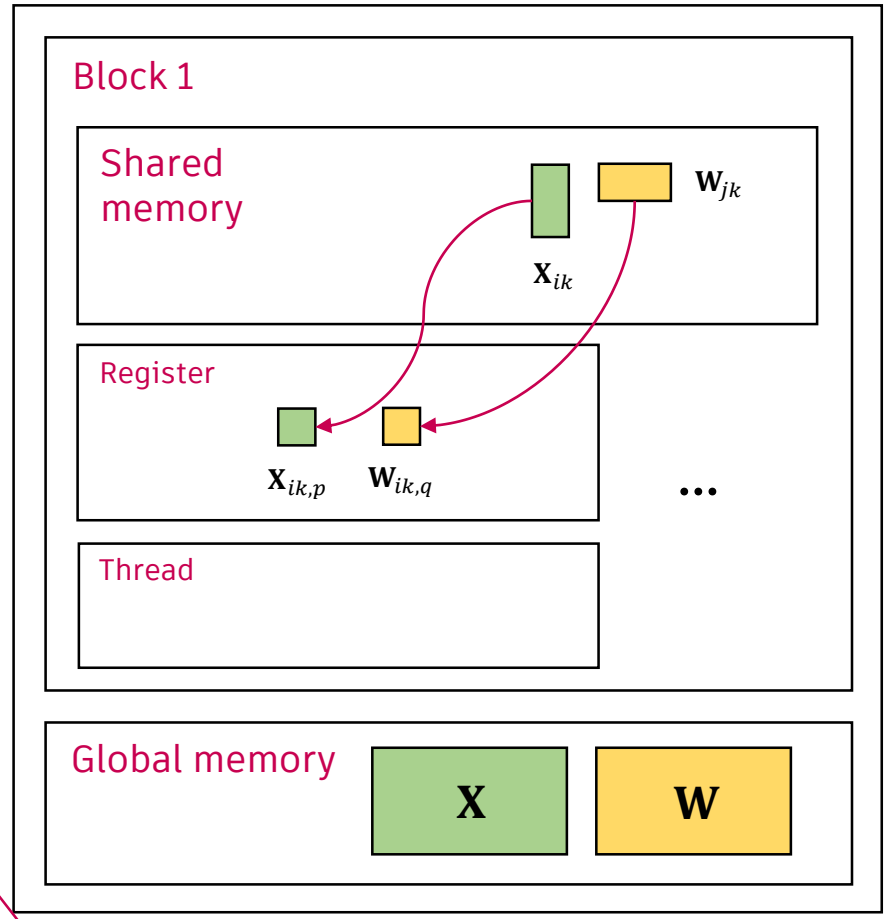
GPU memory hierarchy

4 Method Linear Layer Operator

Level 2: Quantization block



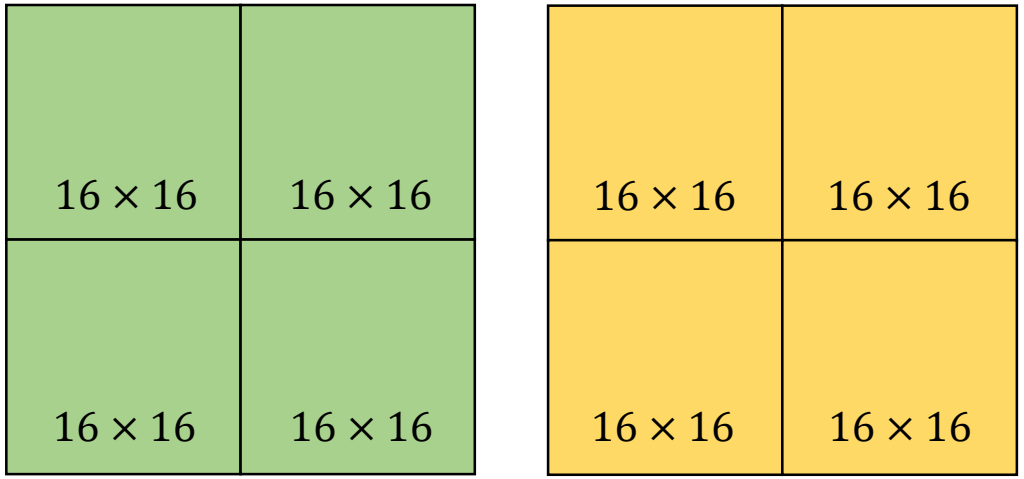
Level 3



GPU memory hierarchy

4 Method Linear Layer Operator

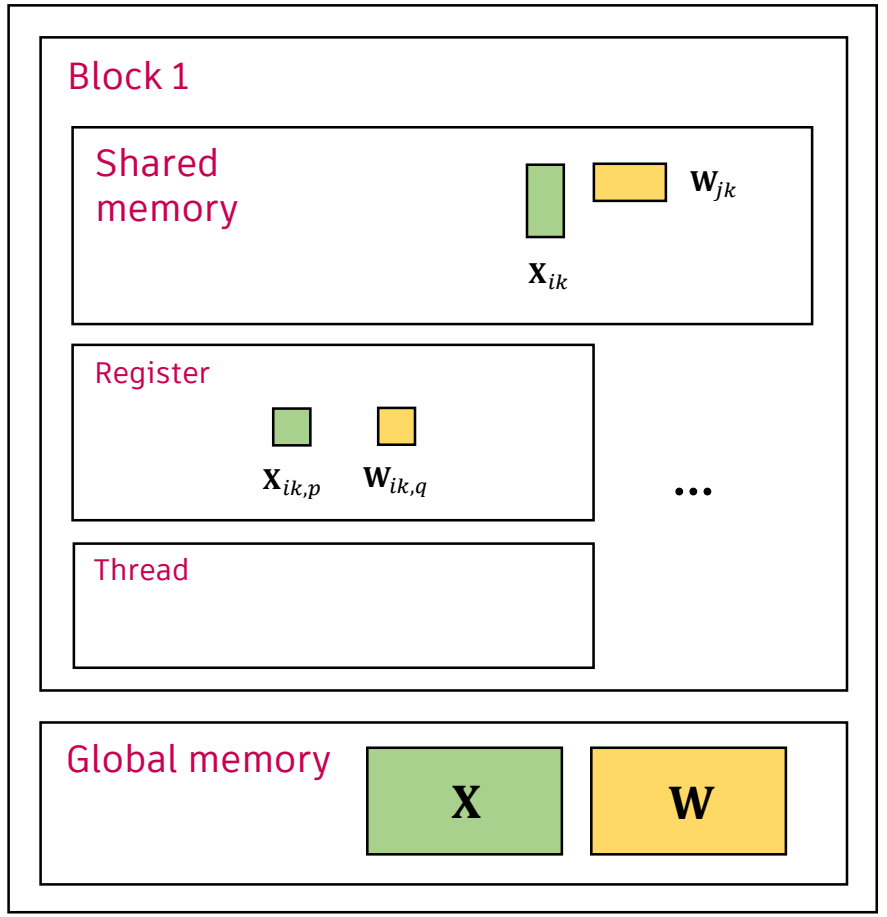
Level 3: WMMA operation



$\mathbf{X}_{ik,p}$ $\mathbf{W}_{jk,q}$

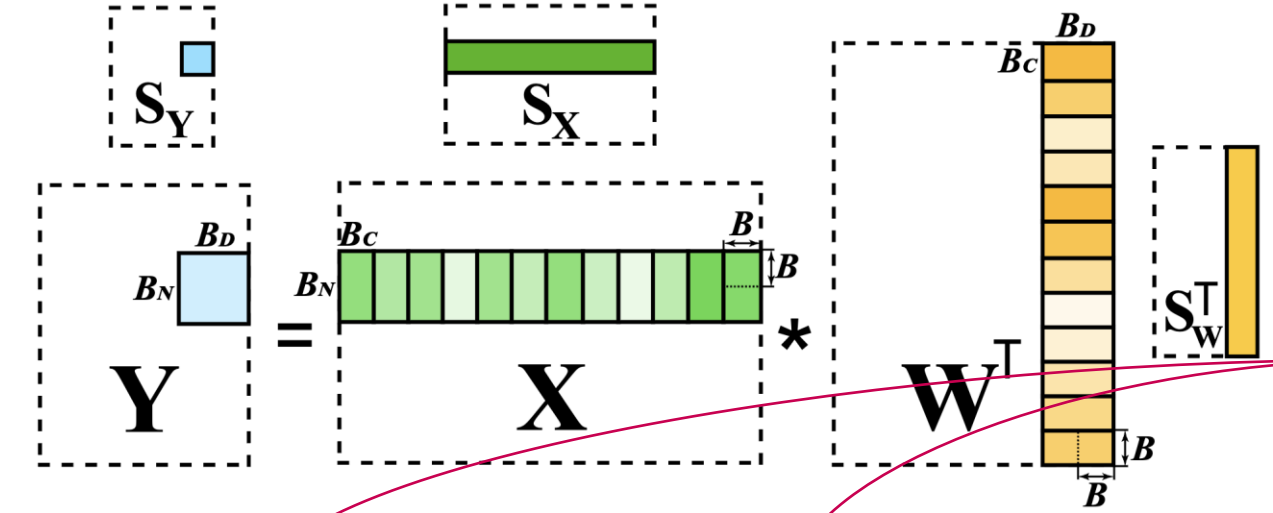
$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} = \begin{bmatrix} AA' + BC' & AB' + BD' \\ A'C + C'D & B'C + DD' \end{bmatrix}$$

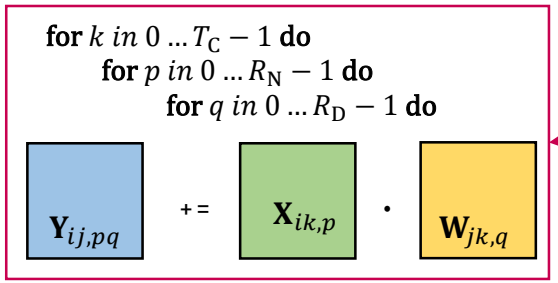
8 x WMMA



GPU memory hierarchy

4 Method Linear Layer Operator





Algorithm 1 INT8 Linear Layer

Require: INT8 Matrices $\mathbf{X} \in \mathbb{R}^{N \times C}$, $\mathbf{W} \in \mathbb{R}^{D \times C}$, FP16 scale matrices $\mathbf{S_X} \in \mathbb{R}^{L_N \times L_C}$, $\mathbf{S_W} \in \mathbb{R}^{L_D \times L_C}$, CUDA Block size $B_N \times B_C \times B_D$

Define $T_N = \left\lceil \frac{N}{B_N} \right\rceil, T_C = \left\lceil \frac{C}{B_C} \right\rceil, T_D = \left\lceil \frac{D}{B_D} \right\rceil$

Define $R_N = \left\lceil \frac{B_N}{B} \right\rceil, R_C = \left\lceil \frac{B_C}{B} \right\rceil R_D = \left\lceil \frac{B_D}{B} \right\rceil$

for $1 \leq \mathrm{i} \leq T_N$ **do**

  for $1 \leq \mathrm{j} \leq T_D$ **do**

    Initialize accumulator $\mathbf{Y}_{ij}^{\mathrm{FP32}}$, $\mathbf{Y}_{ij}^{\mathrm{INT32}}$

    for $1 \leq \mathrm{k} \leq T_C$ **do**

      Load INT8 Block $\mathbf{X}_{ik} \in \mathbb{R}^{B_N \times B_C}$ and scale factor $\mathbf{S}_{\mathbf{X}_{ik}} \in \mathbb{R}^{R_N \times R_C}$

      Load INT8 Block $\mathbf{W}_{jk}^{\top} \in \mathbb{R}^{B_C \times B_D}$ and scale factor $\mathbf{S}_{\mathbf{W}_{jk}^{\top}} \in \mathbb{R}^{R_C \times R_D}$

      On chip, compute INT8 Matmul: $\mathbf{Y}_{ij}^{\mathrm{INT32}} = \mathbf{X}_{ik}\mathbf{W}_{jk}^{\top}$

      On chip, dequantize to FP32 and accumulate: $\mathbf{Y}_{ij}^{\mathrm{FP32}} \leftarrow \mathbf{Y}_{ij}^{\mathrm{FP32}} + \mathbf{S}_{\mathbf{X}_{ik}} \mathbf{Y}_{ij}^{\mathrm{INT32}} \mathbf{S}_{\mathbf{W}_{jk}^{\top}}$

    end for

    On Chip, quantize the output $\mathbf{Y}_{ij}^{\mathrm{FP32}}$ to get $\mathbf{Y}_{ij}^{\mathrm{INT8}} \in \mathbb{R}^{B_N \times B_D}$ and scale $\mathbf{S}_{ij} \in \mathbb{R}^{R_N \times R_D}$

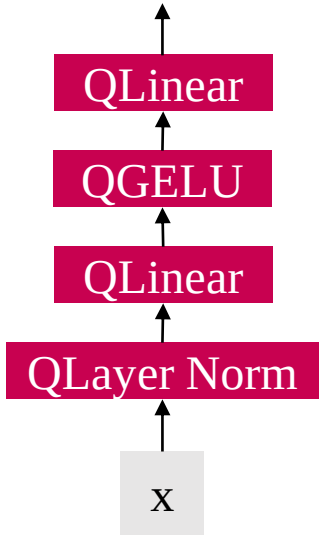
    Save $\mathbf{Y}_{ij}^{\mathrm{INT8}}$ and $\mathbf{S}_{\mathbf{Y}_{ij}}$ to global memory.

  end for

end for

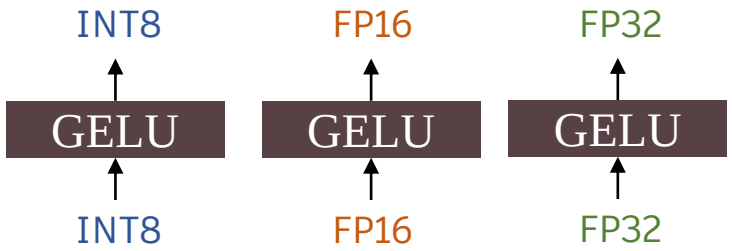
4 Method

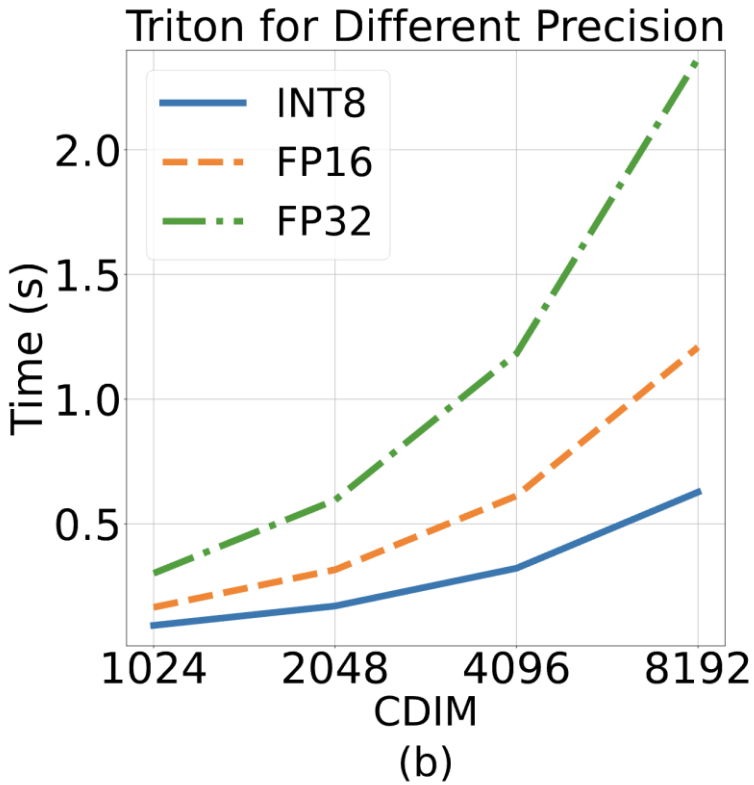
1) Per Block Quantization
2) Linear Layer Operator
3) **Non-Linear Layer Operator**



4 Method Non-Linear Layer Operator

Authors observed that non-linear operation are memory-bounded

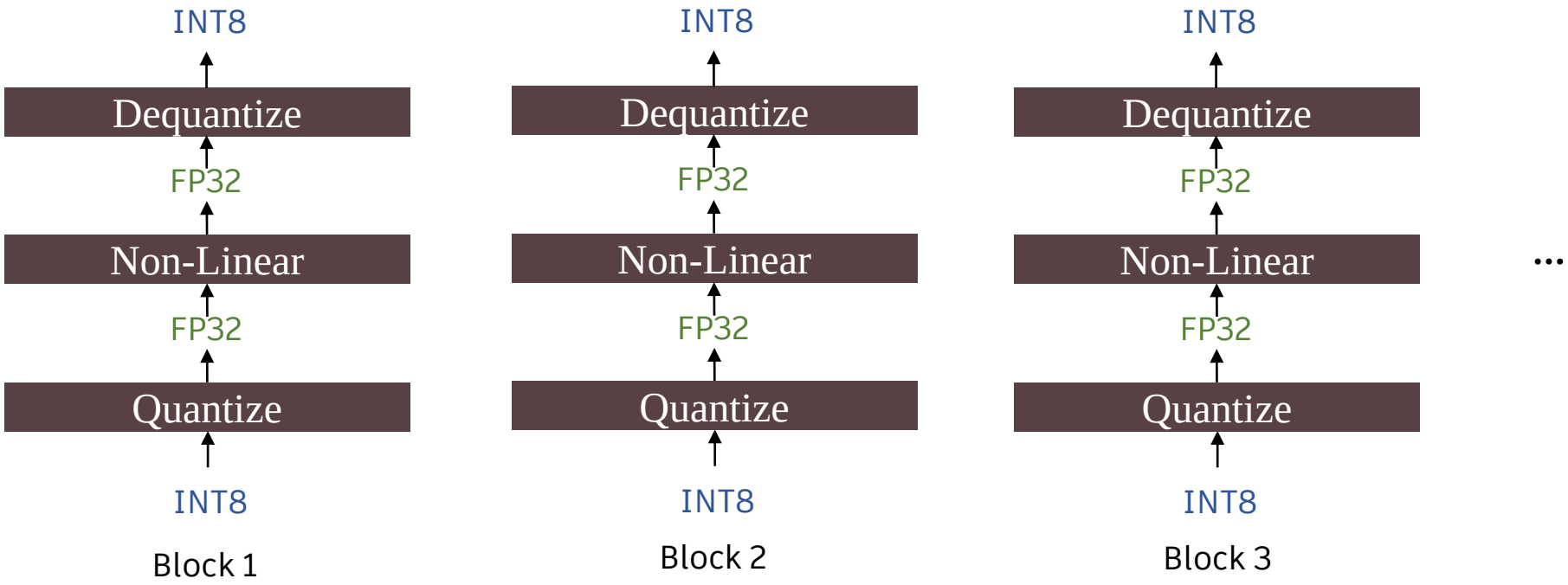




Precision differences in global memory read and write operations on GLEU

4 Method Non-Linear Layer Operator

GELU, LayerNorm, Dropout, Add



All FP32 values are stored only in shared memory due to kernel fusion, while global memory handles only INT8 read/write operations

5 Experimental Settings

- Task
  - Machine translation, image classification, generative model pretraining

- Quantization
  - INT 8: Linear layers of MLP, attention, non-linear layers (GELU, LayerNorm, Dropout)
  - FP 16: Multi-head attention (used FlashAttention[1])
  - FP 32: Master copy of the weights

- Comparison method
  - FP16: Floating point training baseline
  - Per-tensor quantization
  - SwitchBack

[1] Dao, T., Fu, D., Ermon, S., Rudra, A., and R´e, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in Neural Information Process- ing Systems, 35:16344–16359, 2022.

6 Experimental Results

Converged model accuracy

- Machine translation - transformer
  - Train a transformer base model on WMT 14 En-De dataset

Table 4. Results on machine translation, deit pretraining, GPT2 pretraining, and GLUE fine-tuning result based on the pretrained model. FP refers to floating-point, SwitchBack refers to per-token quantization. '–' means the model does not converge.

			Baseline			Ours
Model	#Params(M)	Metric	FP	SwitchBack	Per-Tensor	Jetfire
Transformer-Base	61	BLEU	26.49	26.46	26.04	**26.49**
DeiT-tiny	5	Top1 Acc	64.08	63.55	–	**63.95**
DeiT-small	22		73.43	72.80	–	**73.31**
DeiT-base	86		75.67	75.62	–	**76.03**
GPT2-base	124	Valid loss	2.9074	3.0796	3.1638	**2.8597**
GPT2-medium	350		2.6612	2.9141	3.1795	**2.4195**
GPT2-large	774		2.5993	3.0512	2.9775	**2.4696**
GPT2-base	124	GLUE score	$78.50_{0.45}$	$78.15_{0.15}$	$76.33_{0.94}$	$\mathbf{78.18_{0.34}}$
GPT2-medium	350		$82.00_{0.50}$	$80.03_{0.15}$	$79.19_{0.22}$	$\mathbf{81.60_{0.26}}$
GPT2-large	774		$83.01_{0.24}$	$78.74_{0.24}$	$75.88_{0.35}$	$\mathbf{82.94_{0.70}}$

6 Experimental Results

Converged model accuracy

- Image classification – Deit
  - Pretraining for Deit (Tiny, Small, Base) model on ImageNet1K

Table 4. Results on machine translation, deit pretraining, GPT2 pretraining, and GLUE fine-tuning result based on the pretrained model. FP refers to floating-point, SwitchBack refers to per-token quantization. '–' means the model does not converge.

			Baseline			Ours
Model	#Params(M)	Metric	FP	SwitchBack	Per-Tensor	Jetfire
Transformer-Base	61	BLEU	26.49	26.46	26.04	**26.49**
Deit-tiny	5	Top1 Acc	64.08	63.55	–	**63.95**
Deit-small	22		73.43	72.80	–	**73.31**
Deit-base	86		75.67	75.62	–	**76.03**
GPT2-base	124	Valid loss	2.9074	3.0796	3.1638	**2.8597**
GPT2-medium	350		2.6612	2.9141	3.1795	**2.4195**
GPT2-large	774		2.5993	3.0512	2.9775	**2.4696**
GPT2-base	124	GLUE score	$78.50_{0.45}$	$78.15_{0.15}$	$76.33_{0.94}$	$\mathbf{78.18_{0.34}}$
GPT2-medium	350		$82.00_{0.50}$	$80.03_{0.15}$	$79.19_{0.22}$	$\mathbf{81.60_{0.26}}$
GPT2-large	774		$83.01_{0.24}$	$78.74_{0.24}$	$75.88_{0.35}$	$\mathbf{82.94_{0.70}}$

6 Experimental Results

Converged model accuracy

- Image classification – Swin transformer and ViT
  - Pretraining for Swin (Tiny, Small, Base) model on ImageNet1K
  - Fine-tune ViT (Base, Large) model on ImageNet1K

MODEL	SWIN-TINY	SWIN-SMALL	SWIN-BASE
FP	77.55	80.39	80.45
JETFIRE	77.51	80.39	80.37
	VIT-BASE	VIT-LARGE	
FP	83.45	85.72	
JETFIRE	83.48	85.67	

Table 5. Comparison of FP and Jetfire

6 Experimental Results

Converged model accuracy

- Generative model pretraining– GPT2
  - Training for GPT2 (base, medium, large) model on OpenWebText

Table 4. Results on machine translation, deit pretraining, GPT2 pretraining, and GLUE fine-tuning result based on the pretrained model. FP refers to floating-point, SwitchBack refers to per-token quantization. '–' means the model does not converge.

			Baseline			Ours
Model	#Params(M)	Metric	FP	SwitchBack	Per-Tensor	Jetfire
Transformer-Base	61	BLEU	26.49	26.46	26.04	**26.49**
DeiT-tiny	5	Top1 Acc	64.08	63.55	–	**63.95**
DeiT-small	22		73.43	72.80	–	**73.31**
DeiT-base	86		75.67	75.62	–	**76.03**
GPT2-base	124	Valid loss	2.9074	3.0796	3.1638	**2.8597**
GPT2-medium	350		2.6612	2.9141	3.1795	**2.4195**
GPT2-large	774		2.5993	3.0512	2.9775	**2.4696**
GPT2-base	124	GLUE score	$78.50_{0.45}$	$78.15_{0.15}$	$76.33_{0.94}$	$\mathbf{78.18_{0.34}}$
GPT2-medium	350		$82.00_{0.50}$	$80.03_{0.15}$	$79.19_{0.22}$	$\mathbf{81.60_{0.26}}$
GPT2-large	774		$83.01_{0.24}$	$78.74_{0.24}$	$75.88_{0.35}$	$\mathbf{82.94_{0.70}}$

7 Ablation Study

CUDA kernel and Triton kernel block size

- Block size for Triton and CUDA kernels is crucial
  - Large: Decrease in parallelism
  - Small: Low utilization of bandwidth and computational resources

- Block size of Triton: 64×64
- Block size of CUDA: $128 \times 32 \times 128$

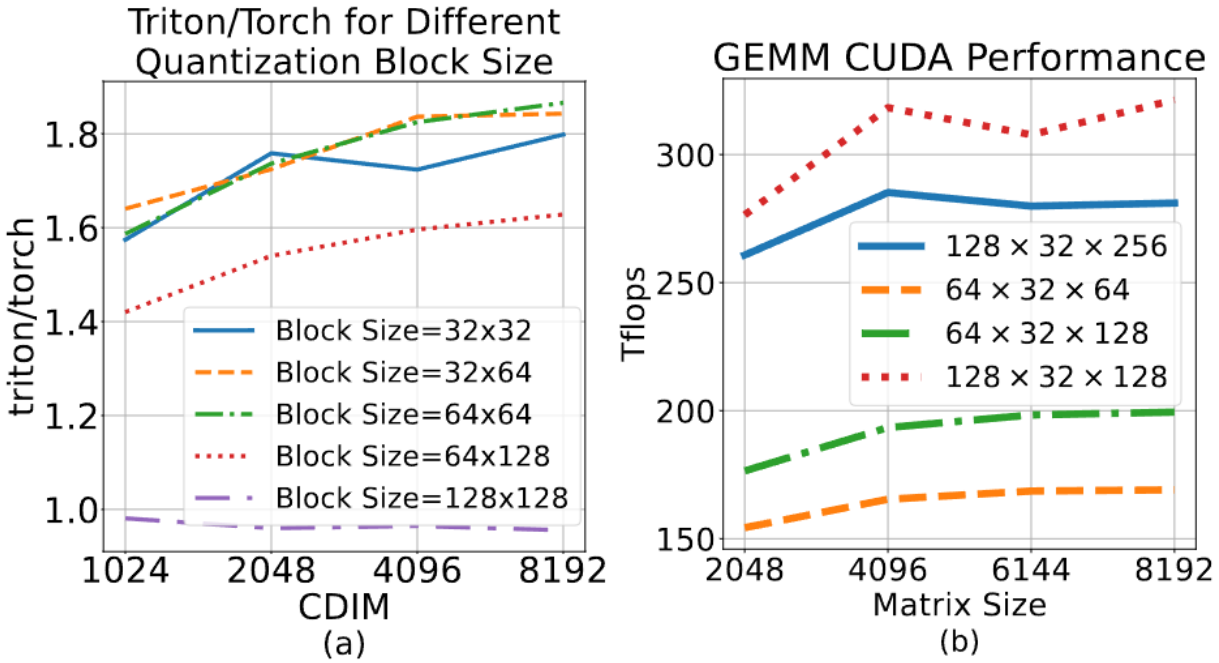


Figure 6. Speed test of GELU and GEMM operator. (a) Triton kernel speedup with different Triton block sizes.(b) GEMM CUDA kernel speed with different CUDA block sizes.

8 Operator and End-to-End Experiments

Linear layer & non-linear operator speedup

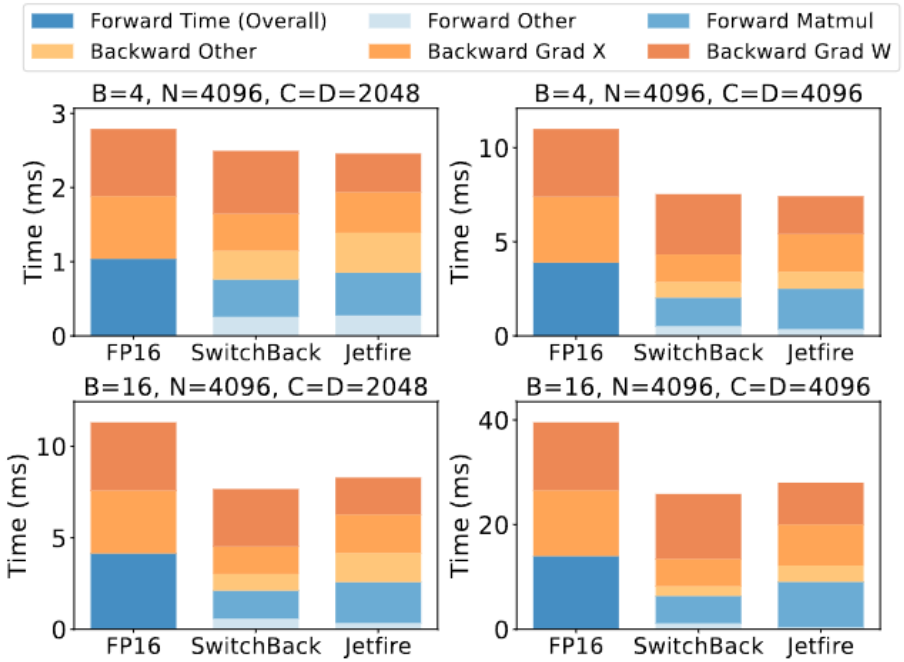


Figure 8. Matrix Multiplication Speed test for Different Methods in different settings (B=Batch Size, N=Sequence Length).

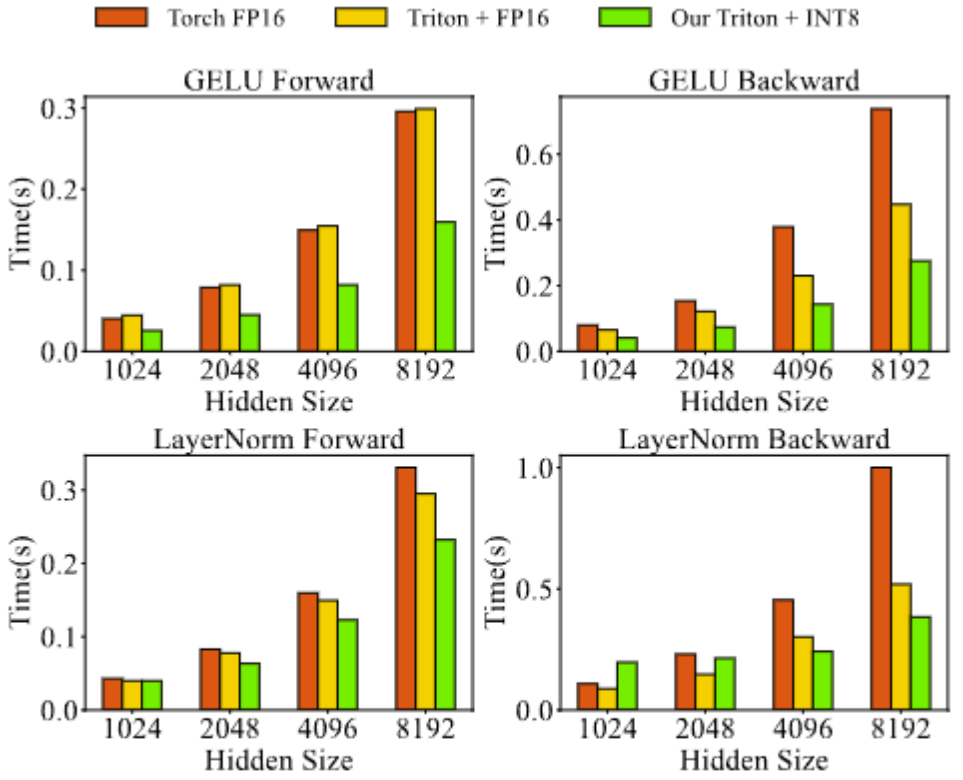


Figure 7. Speed comparision between our INT8 non-linear operator and pytorch FP16 implementation.

8 Operator and End-to-End Experiments

End-to-end speedup

- Show the end-to-end speedup for Jetfire method over PyTorch's FP16

Table 6. Acceleration ratios for End-to-end comparison (SB refers to SwitchBack basic version) on GPT2 model.

	FORWARD		BACKWARD		OVERALL	
HIDDEN SIZE	SB	OURS	SB	OURS	SB	OURS
4096	1.50	1.32	1.18	1.46	1.27	**1.42**
2048	1.53	1.29	1.24	1.41	1.32	**1.37**
1024	0.94	0.97	1.14	1.11	**1.07**	**1.07**

9 Conclusion

- Jetfire proves that full-pipeline INT8 pre-training can match FP baselines without accuracy loss

- By storing activations, weights, and gradients in INT8, it halves computation, memory traffic, and GPU usage

10 Limitation

- Multi-Head Attention remains in FP16 via the FlashAttention kernel, so Jetfire is not a fully INT8 end-to-end pipeline
- The experiments lack INT4 Transformer[2] baselines
- The per-block quantization scale matrix is large, leading to increased communication overhead for scale values.
- 4-bit per block quantization may not be accurately performed in a Transformer.

[2] Xi, H., Li, C., Chen, J., & Zhu, J. (2023). Training transformers with 4-bit integers. Advances in Neural Information Processing Systems, 36, 49146-49168.

Thank You!

Appendix

Table 2. Meaning of Key Constants.

$B_N/B_C/B_D$	CUDA block size in MM
$T_N/T_C/T_D$	Number of CUDA blocks along each axis
B	Quantization block size
$R_N/R_C/R_D$	Number of quantization blocks in a CUDA block along each axis

Table 3. Time complexity of different operations in MM.

	Method		
Operation	Basic INT8	SwitchBack	Ours
MM	$B_N B_D C$	$B_N B_D C$	$B_N B_D C$
16-bit Load/Store	$(B_N + B_D)C + B_N B_D$	$\frac{B_N B_C}{T_D} + \frac{B_D B_C}{T_N} + B_N B_D$	-
8-bit Load/Store	-	$(B_N + B_D)C$	$(B_N + B_D)C + B_N B_D$
Dequantize	-	$B_N B_D$	$B_N B_D T_C$
Quantize	-	$\frac{B_N B_C}{T_D} + \frac{B_D B_C}{T_N}$	$B_N B_D$

Appendix

Algorithm 2 INT8 Non-Linear Operator

Require: INT8 Matrix $\mathbf{X} \in \mathbb{R}^{N \times C}$, FP16 scale matrix $\mathbf{S_X} \in \mathbb{R}^{L_N \times L_C}$, element-wise function f

Define $T_N = \left\lceil \frac{N}{B_N} \right\rceil, T_C = \left\lceil \frac{C}{B_C} \right\rceil$
Define $R_N = \left\lceil \frac{B_N}{B} \right\rceil, R_C = \left\lceil \frac{B_C}{B} \right\rceil$
for $1 \leq \mathrm{i} \leq T_N$ **do**
  for $1 \leq \mathrm{j} \leq T_C$ **do**
    Load INT8 block $\mathbf{X}_{ij} \in \mathbb{R}^{B_N \times B_C}$, $\mathbf{S}_{\mathbf{X}_{ij}} \in \mathbb{R}^{R_N \times R_C}$
    Dequantize $\mathbf{X}_{ij}$ and $\mathbf{S}_{\mathbf{X}_{ij}}$ to get $\mathbf{X}_{ij}^{\mathbf{FP32}}$
    Operate: $\mathbf{Y}_{ij}^{\mathbf{FP32}} = f(\mathbf{X}_{ij}^{\mathbf{FP32}})$
    Quantize $\mathbf{Y}_{ij}^{\mathbf{FP32}}$ to get $\mathbf{Y}_{ij}^{\mathbf{INT32}} \in \mathbb{R}^{B_N \times B_C}$ and scale factor $\mathbf{S}_{\mathbf{Y}_{ij}} \in \mathbb{R}^{R_N \times R_C}$
    Save $\mathbf{Y}_{ij}^{\mathbf{INT32}}$ and $\mathbf{S}_{\mathbf{Y}_{ij}}$ to global memory.
  end for
end for