Quantization of Vision Transformer

MEGVII 旷视

Outline

MEGVII 旷视

- **Background**
  - What is Post Training Quantization?
  - Why int8 is faster on GPU?
  - The introduction of Vision Transformer
- **Related works**
  - Post-Training Quantization for Vision Transformer (NeurIPS-2021)
- **FQ-ViT (this course)**
  - Why fully quantization?
  - Post-Training Quantization for Fully Quantized Vision Transformer (IJCAI-2022)
- **The Sparsebit framework & Homeworks**

Background – What is Post Training Quantization

- **PTQ v.s. QAT**

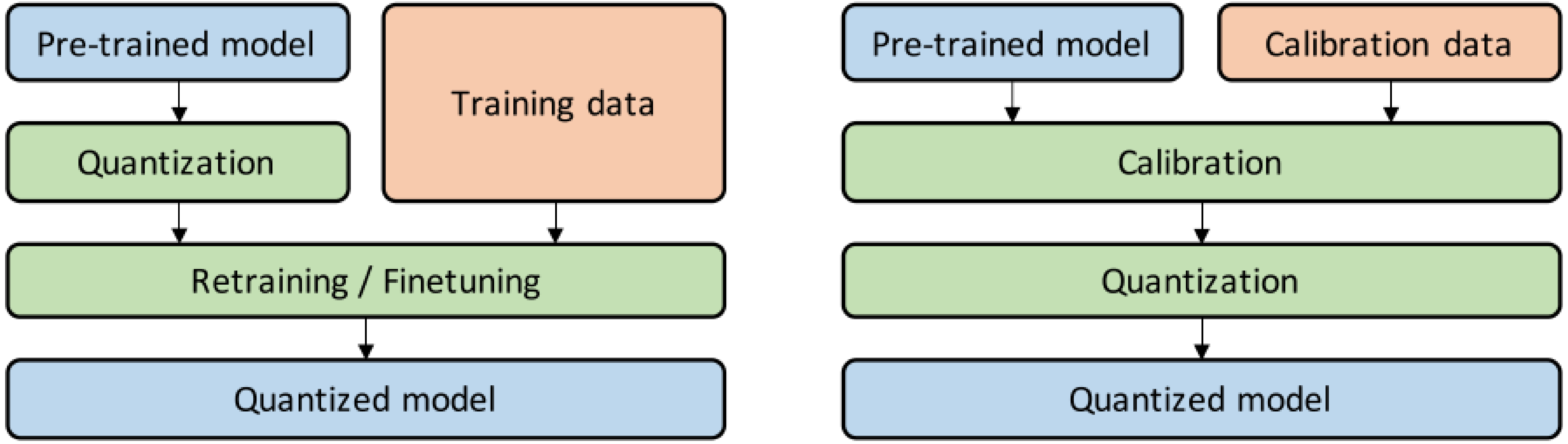


Figure 4: *Comparison between Quantization-Aware Training (QAT, Left) and Post-Training Quantization (PTQ, Right). In QAT, a pre-trained model is quantized and then finetuned using training data to adjust parameters and recover accuracy degradation. In PTQ, a pre-trained model is calibrated using calibration data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors. Then, the model is quantized based on the calibration result. Note that the calibration process is often conducted in parallel with the finetuning process for QAT.*

A. Gholami, S. Kim, et al. A Survey of Quantization Methods for Efficient Neural Network Inference

Background – What is Post Training Quantization

- **Taxonomy**
  - Whether the quantization parameters changed in inference
    - *Static*: the quant params of weights & activations are kept unchanged in inference
    - *Dynamic*: weights are statically quantized, but the quant params of activations changed per-sample
  - With or Without Data
    - *With-Data*: having a subset of training data for calibration
    - Without Data (Data-Free): using synthetic data or metric based for calibration, i.e. ZeroQ, ZAQ...
  - With or Without Finetuning
    - With finetuning: training the quant params or model weights use calibration-set, i.e. AdaRound, BrecQ
    - Without finetuning: use the quant params directly in calibration.

Y. Cai, et al. Zeroq: A novel zero shot quantization framework
Y. Liu, W. Zhang, et al. Zero-shot Adversarial Quantization
M.Negal, et al. AdaRound: Up or Down? Adaptive Rounding for Post-Training Quantization
Y. LI, R, Gong, et al. BRECQ: PUSHING THE LIMIT OF POST-TRAINING QUANTIZATION BY BLOCK RECONSTRUCTION

Background – What is Post Training Quantization

MEGVII 旷视

- **Quantization**

$$x_q = clip\left(round\left(\frac{x}{s}\right), -2^{b-1}, 2^{b-1} - 1\right)$$

$$x_q = clip(round\left(\frac{x}{s}\right) + z, 0, 2^b - 1)$$

- **DeQuantization**

$$\hat{x} = x_q * s$$

$$\hat{x} = \left(x_q - z\right) * s$$

- **MatMul**

$$Y = (X_q W_q) * s_x * s_w$$

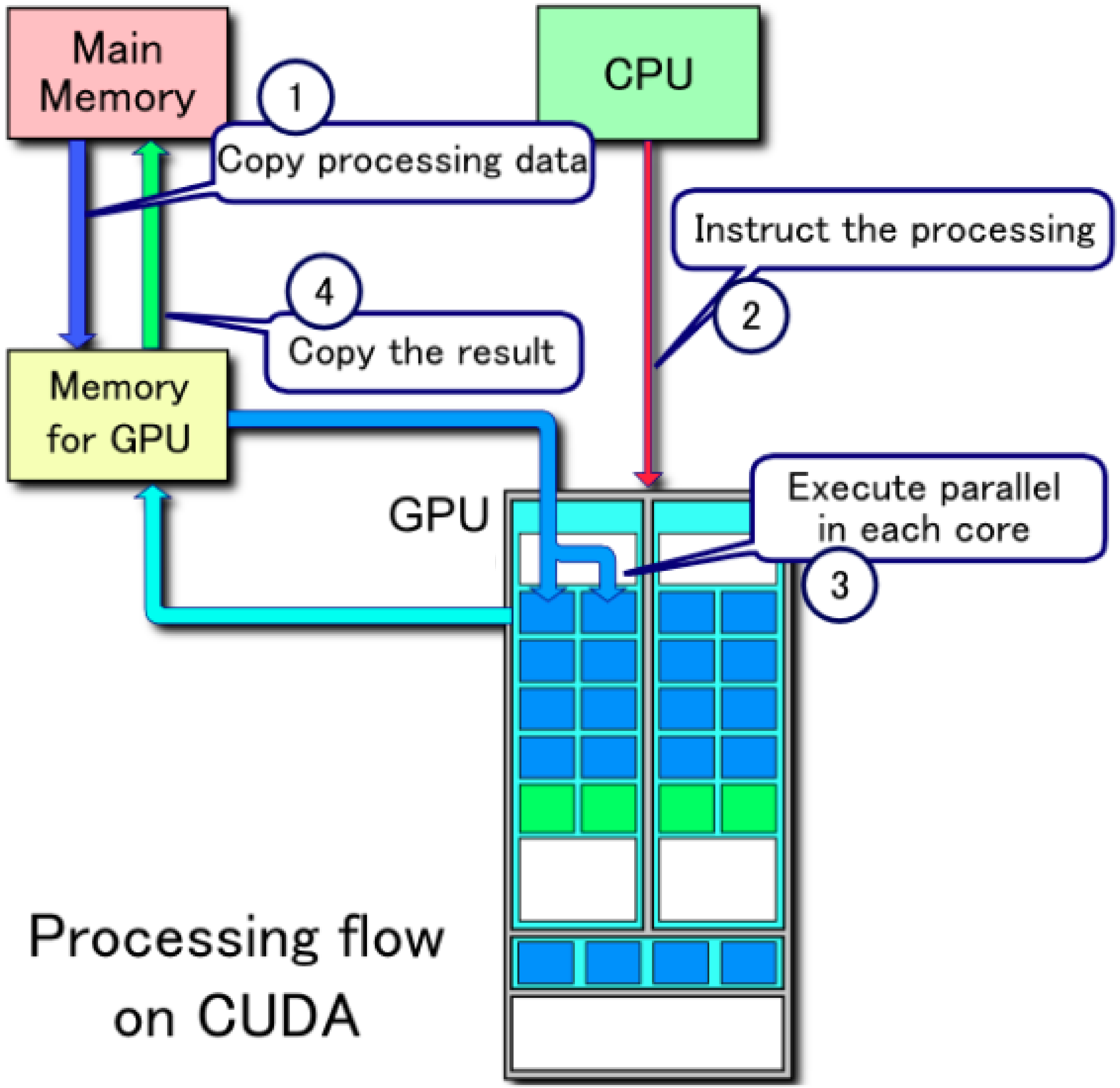
$$Y = \left(x_q - z_x\right) * s_x * \left(w_q - z_w\right) * s_w = s_x s_w (x_q w_q - z_x w_q - z_w x_q - z_w z_x)$$

Background – Why int8 is faster on GPU

MEGVII 旷视

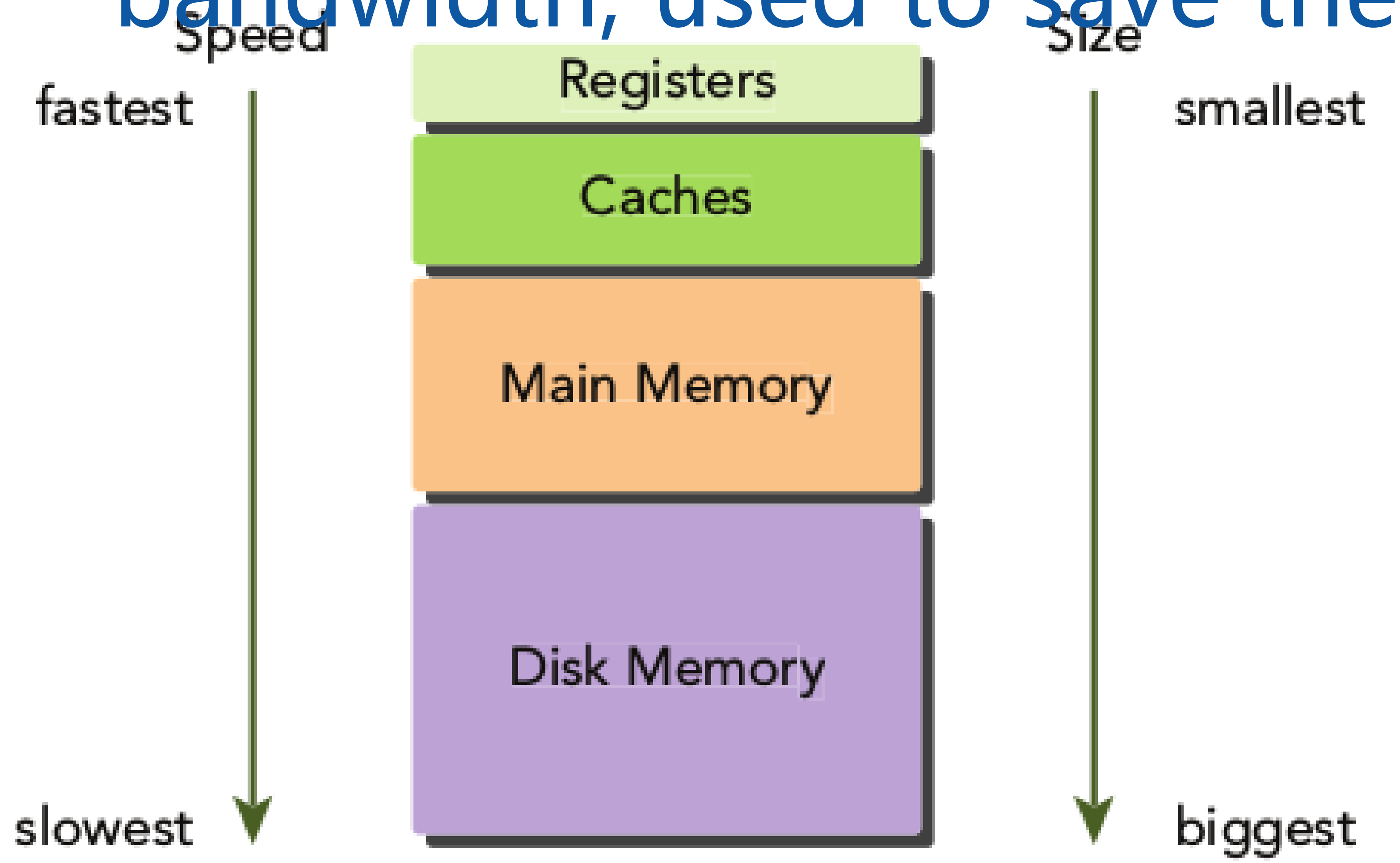
⑩ CUDA Programming Structure

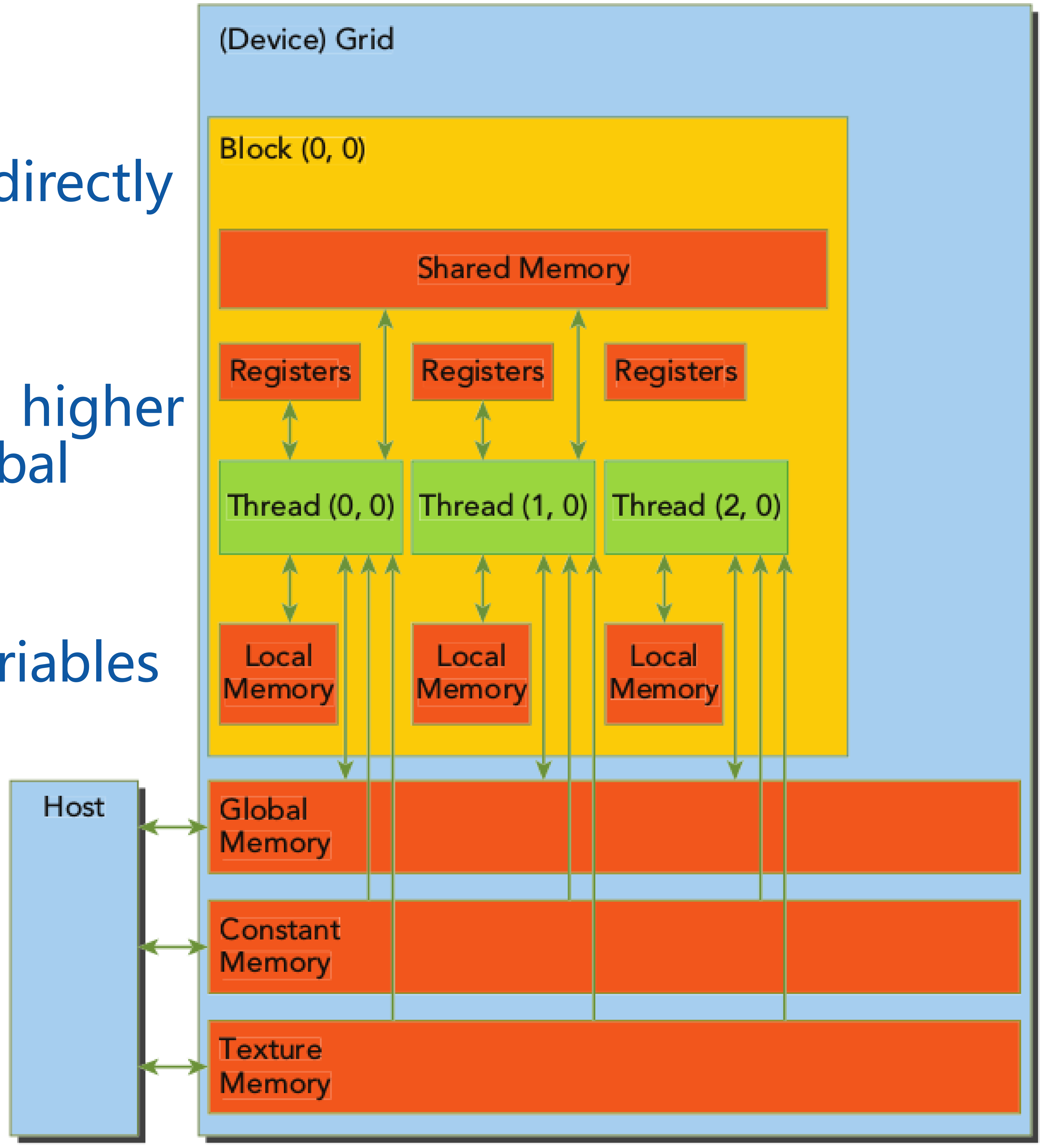
- CPU and GPU (as an external device) form a heterogeneous computing system
- Host and Device: copy data between them before kernel launch and after kernel ending. (i.e. image.cuda(), image.cpu() in pytorch)
- A kernel is a function(or a layer, an operation) which can only be executed on GPU.



Background – Why int8 is faster on GPU

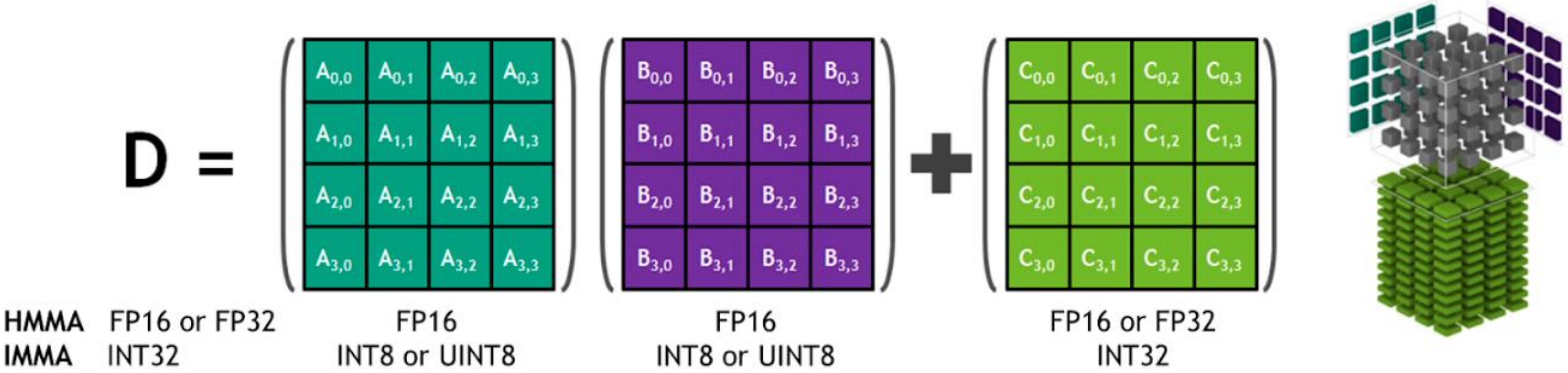
- **Memory structure**
  - **Registers**: fastest, thread-private, not directly programmable
  - **Global Memory**: largest, all-access
  - **Shared Memory**: like L1 cache in CPU, higher bandwidth and lower latency than Global Mem.
  - **Local Memory**: Higher latency, lower bandwidth, used to save the spilled variables





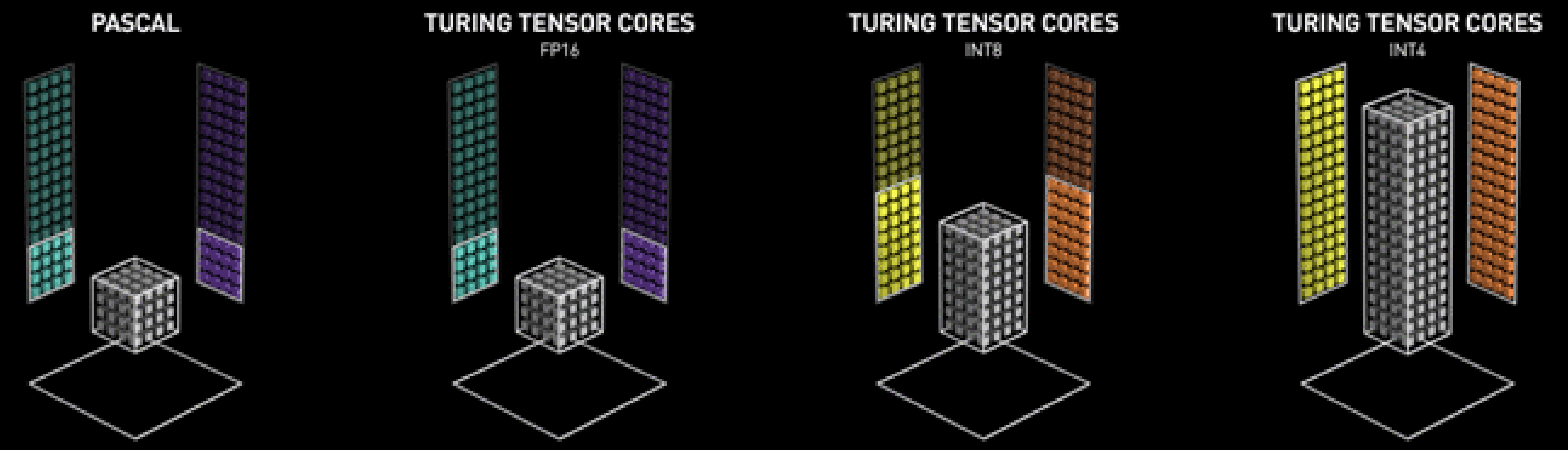
Background – Why int8 is faster on GPU

- **Cuda Core**
  - A standard floating point unit which can execute one operation per clock cycle
  - Less powerful than a desktop CPU core, but when used together for deep learning, many CUDA cores can accelerate computation by executing processes in parallel
- **Tensor Core**
  - The first generation of tensor cores accelerates DL through a fused multiply add computation
  - Allows two 4 x 4 FP16 matrices to be multiplied and added to a 4 x 4 FP16 or FP32 matrix.
  - Also



Background – Why int8 is faster on GPU

MEGVII 旷视

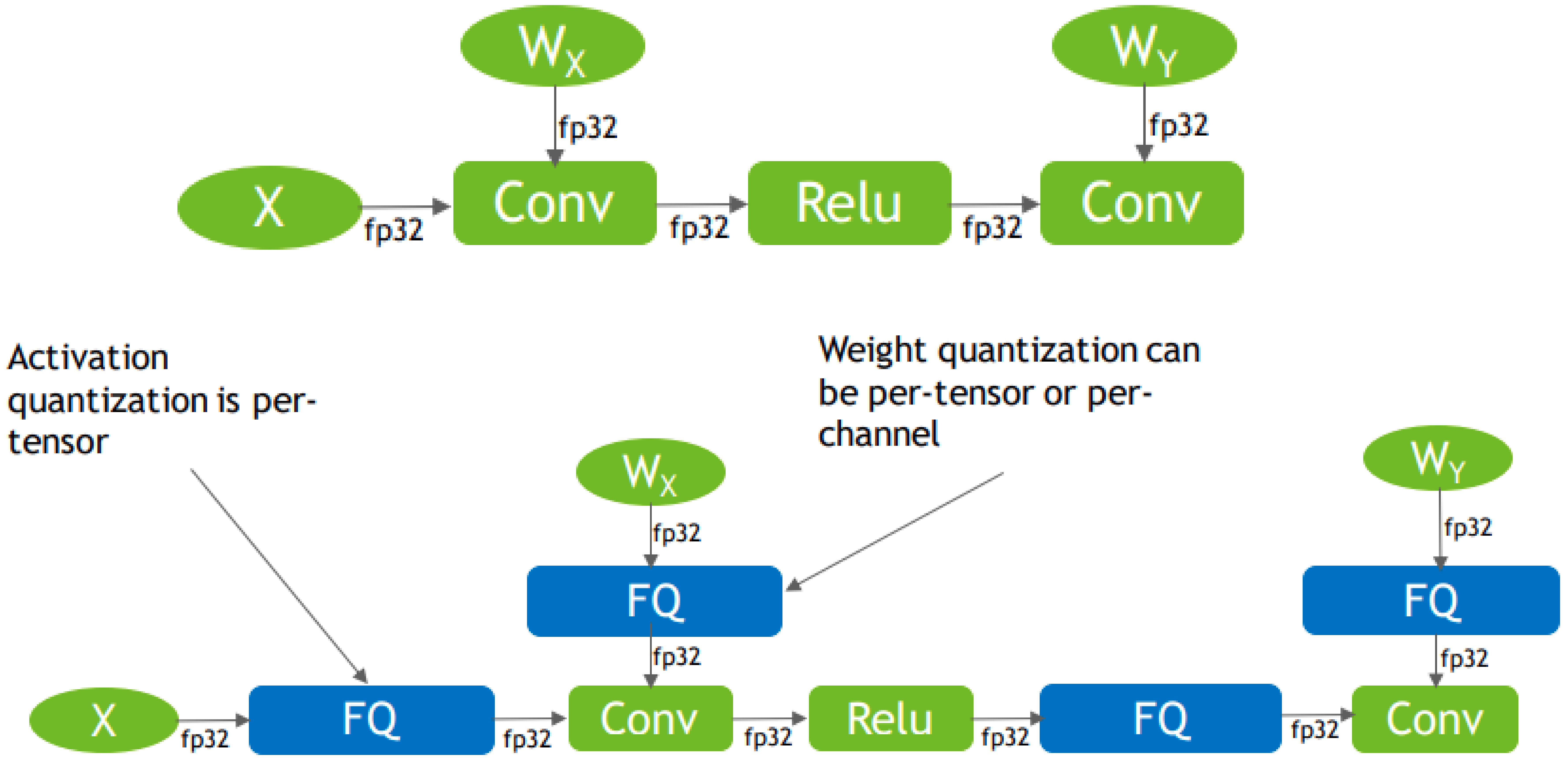


Evolution from Pascal to Turing

Background – Why int8 is faster on GPU

MEGVII 旷视

- **Fake Quant in TensorRT**

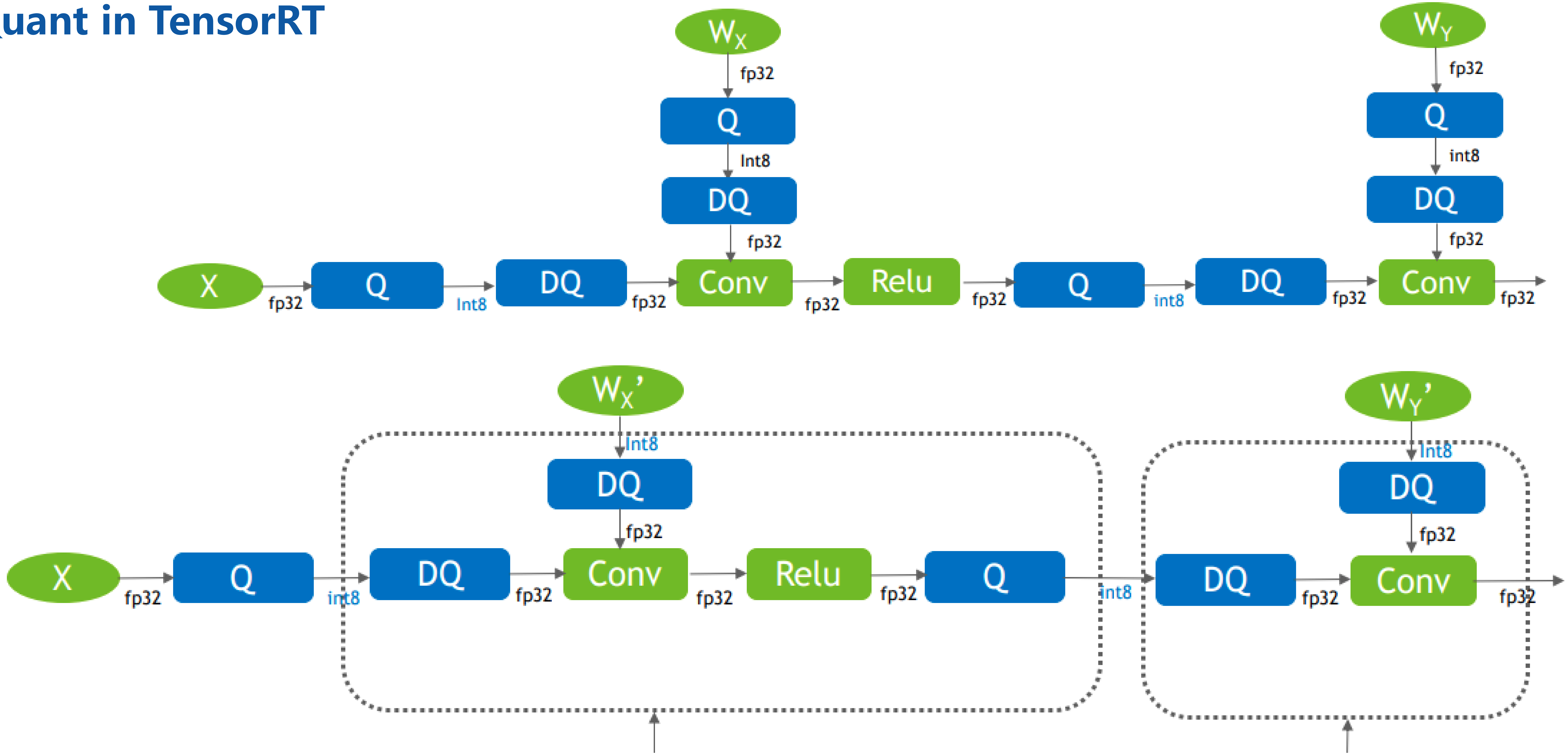


https://developer.download.nvidia.cn/video/gputechconf/gtc/2020/presentations/s21664-toward-int8-inference-deploying-quantization-aware-trained-networks-using-tensorrt.pdf

Background – Why int8 is faster on GPU

MEGVII 旷视

- **Fake Quant in TensorRT**



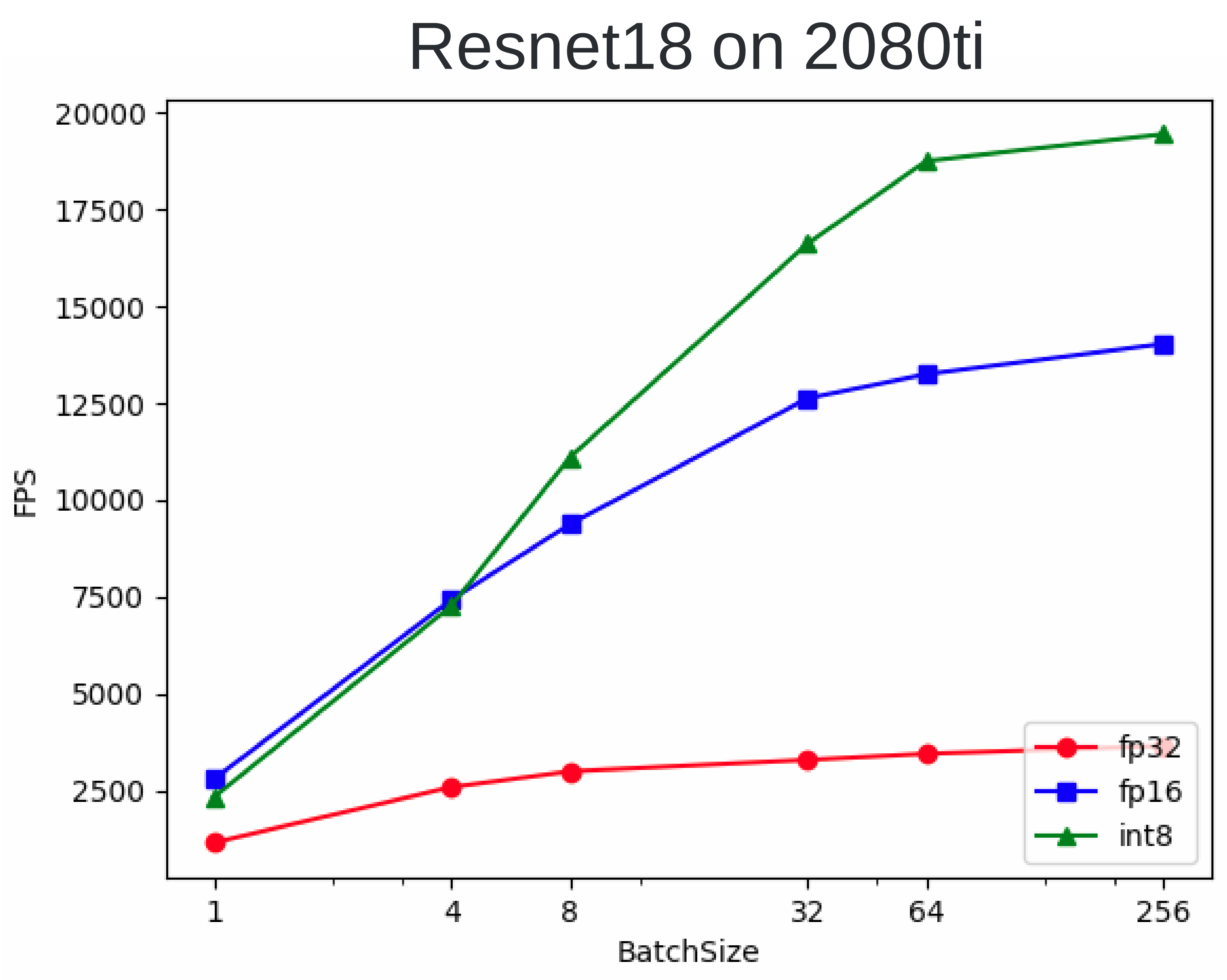
https://developer.download.nvidia.cn/video/gputechconf/gtc/2020/presentations/s21664-toward-int8-inference-deploying-quantization-aware-trained-networks-using-tensorrt.pdf

Background – Why int8 is faster on GPU

MEGVII 旷视

- **Why we need Int8?**
  - Reduce model size so that it can be deployed on resource-constrained devices
  - Reading / writing data from global memory are expensive, often requiring hundreds of cycles.
  - Int8 can use Tensor Cores for GEMM acceleration

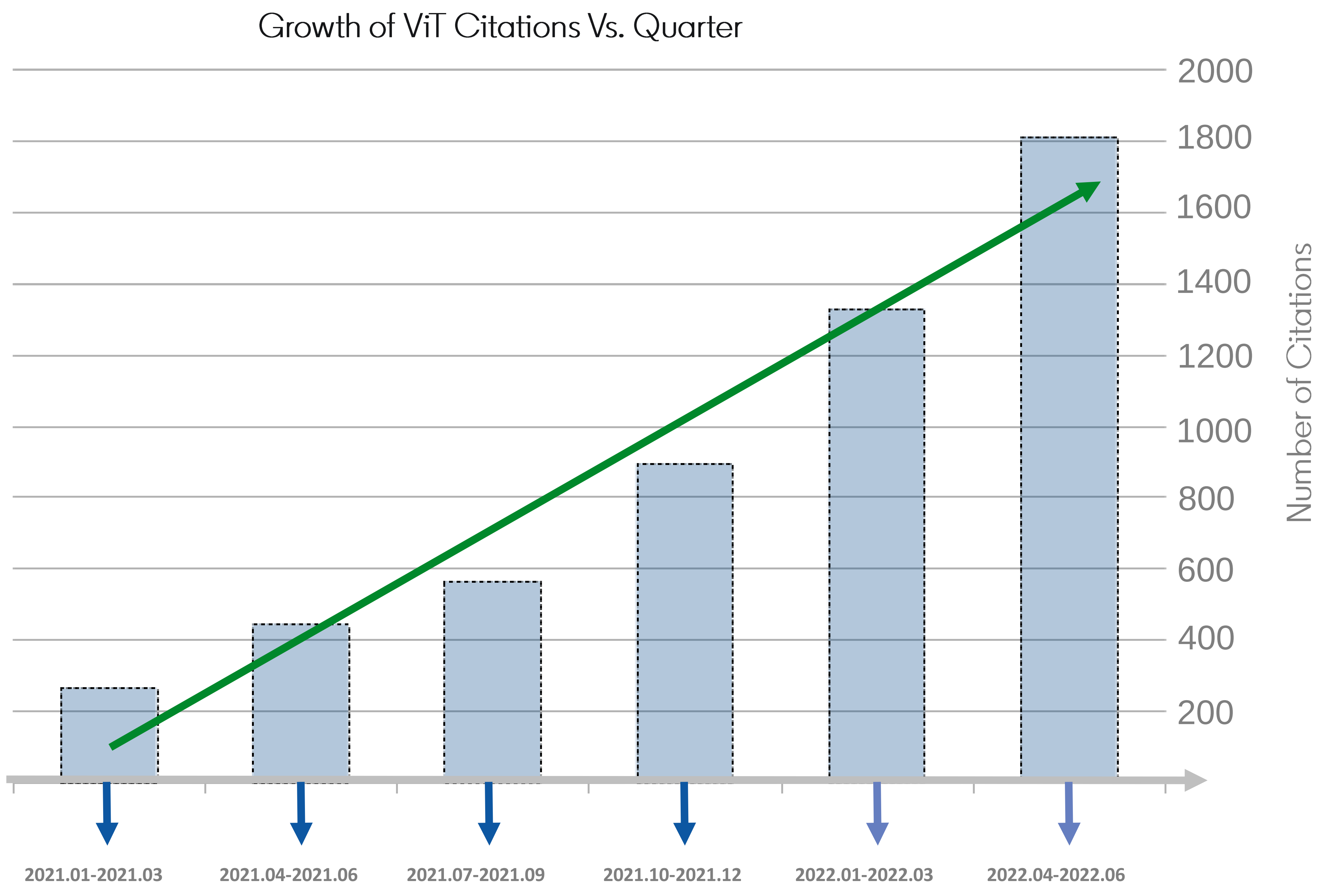
- **FP16 v.s. Int8**
  - When batchsize <= 4, the FPS of fp16 is similar to int8
  - When batchsize increase, the FPS of int8 overtakes fp16
  - When batchsize >=64, the gap between FPS under fp16 and int8 is almost constant.

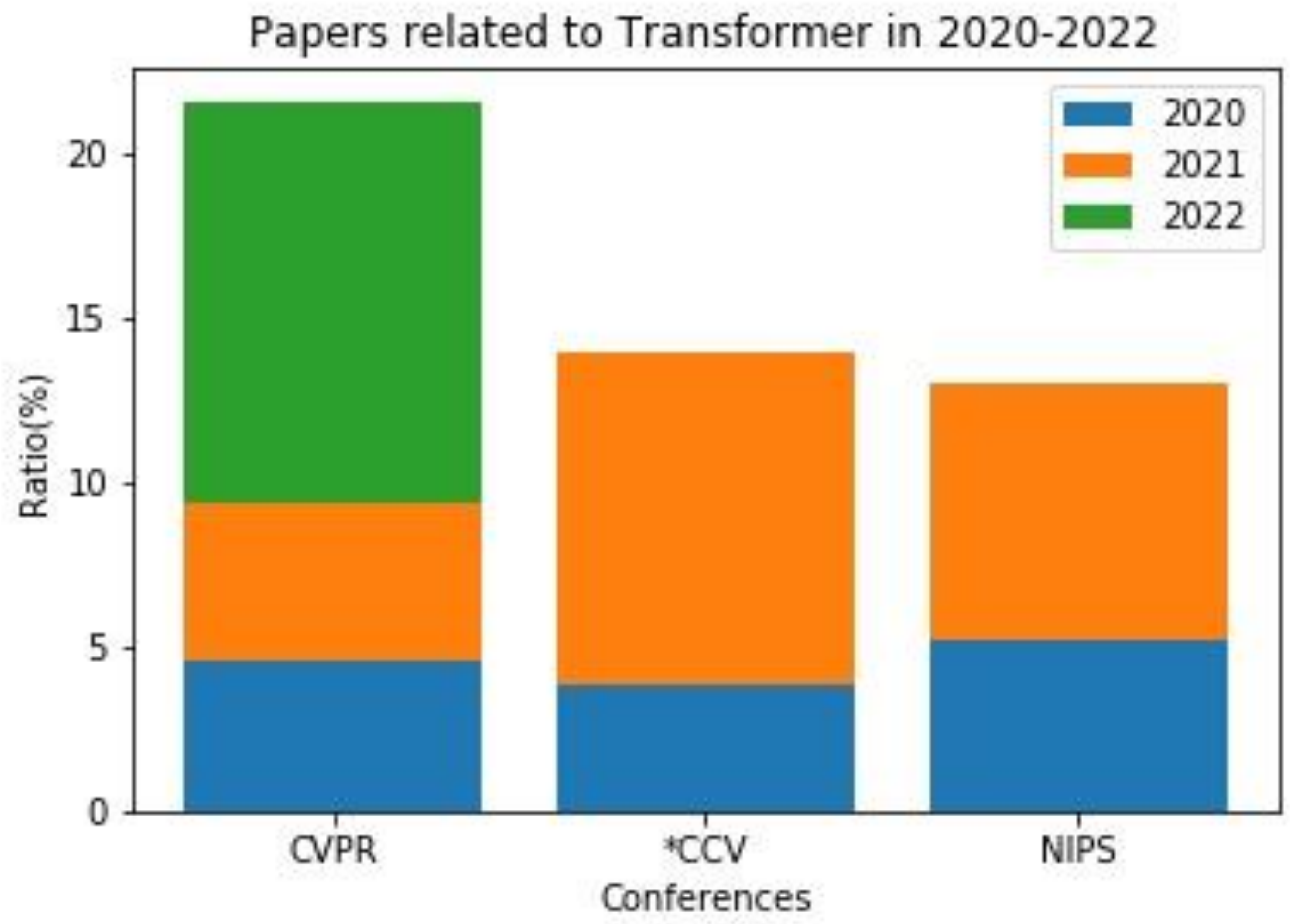


Background – The introduction of Vision Transformer

MEGVII 旷视

- **The trend of Transformers for vision applications**
  - ViT* first published in Oct 2020.





By 07.23.2022: 5575!!!

* A. Dosovitskiy, L. Beyer, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Background – The introduction of Vision Transformer

- **The trend of Vision Transformer on paperwithcode.com**

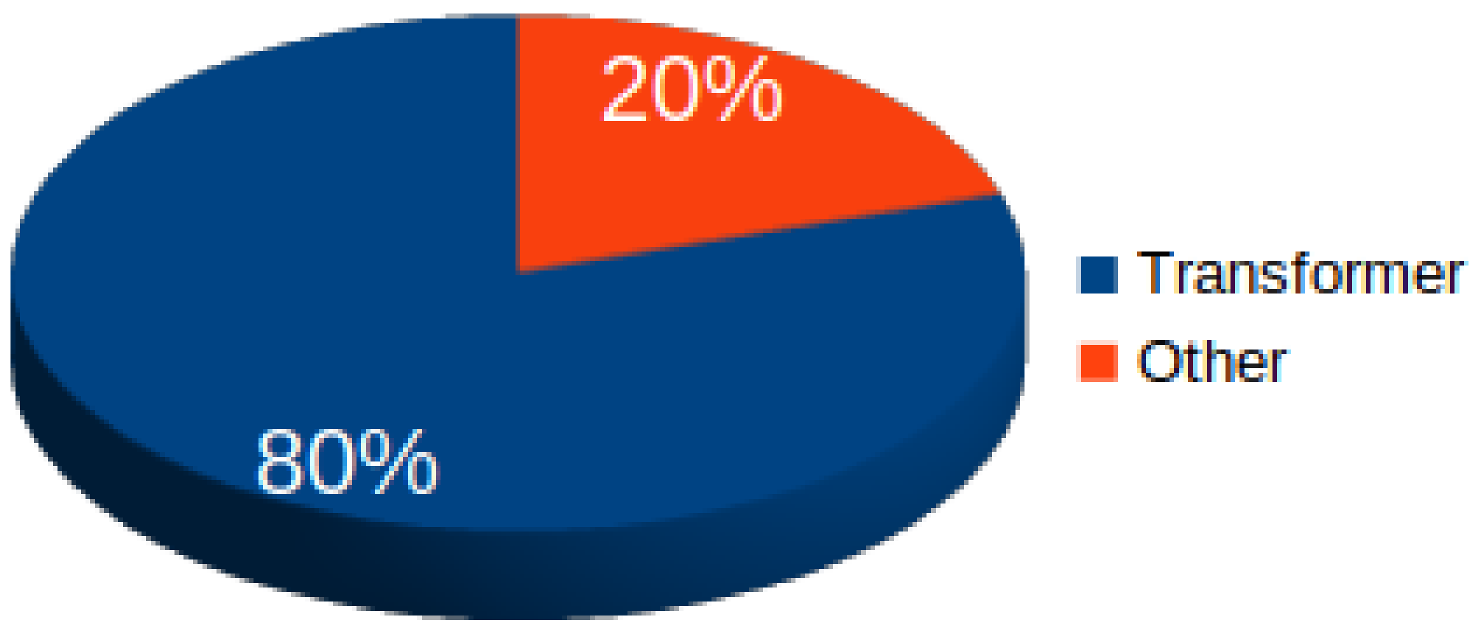
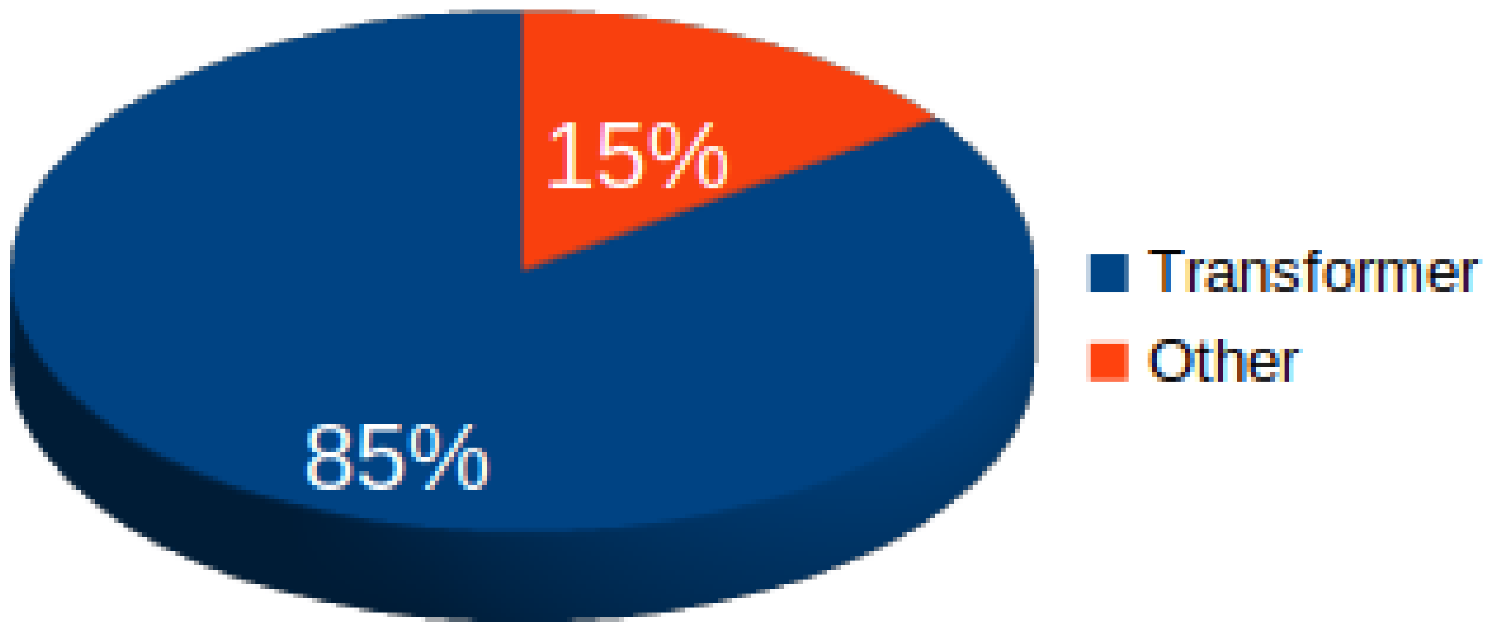


Image classification on ImageNet-1k



Object Detection on COCO

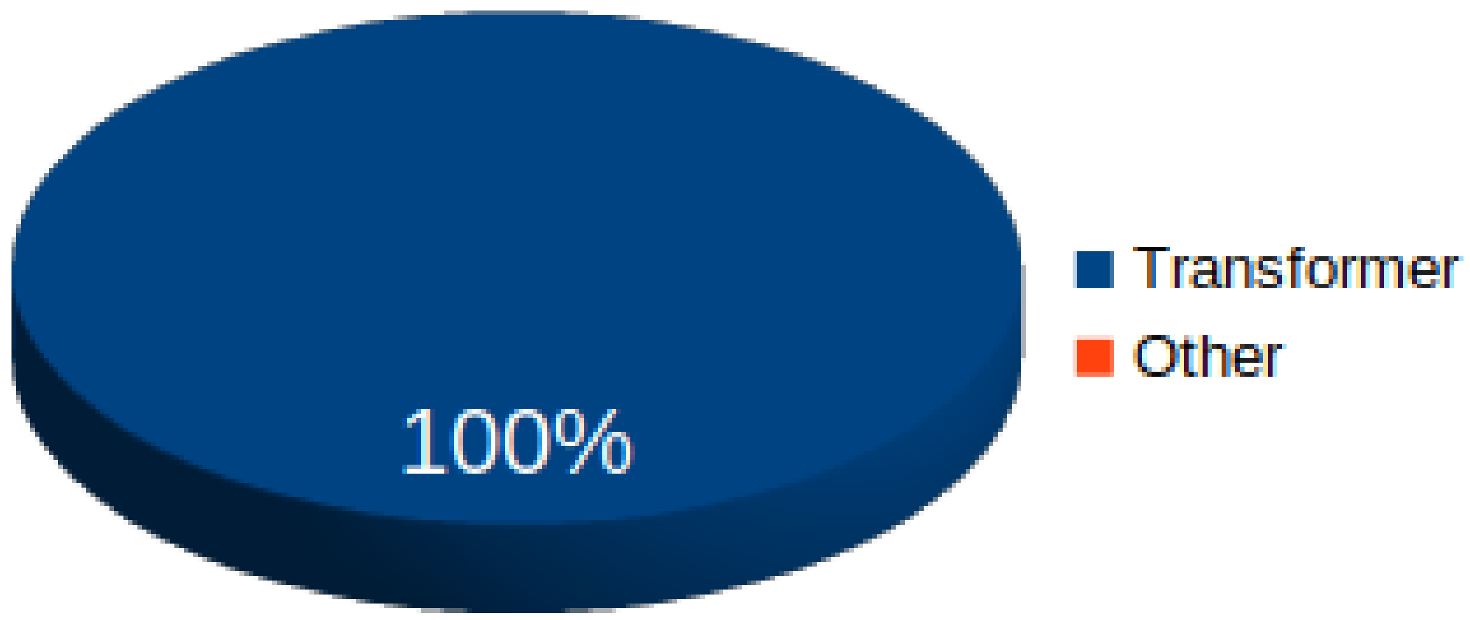


Image Segmentation on ADE-20k

Paper with Code: https://paperswithcode.com/

Background – The introduction of Vision Transformer

MEGVII 旷视

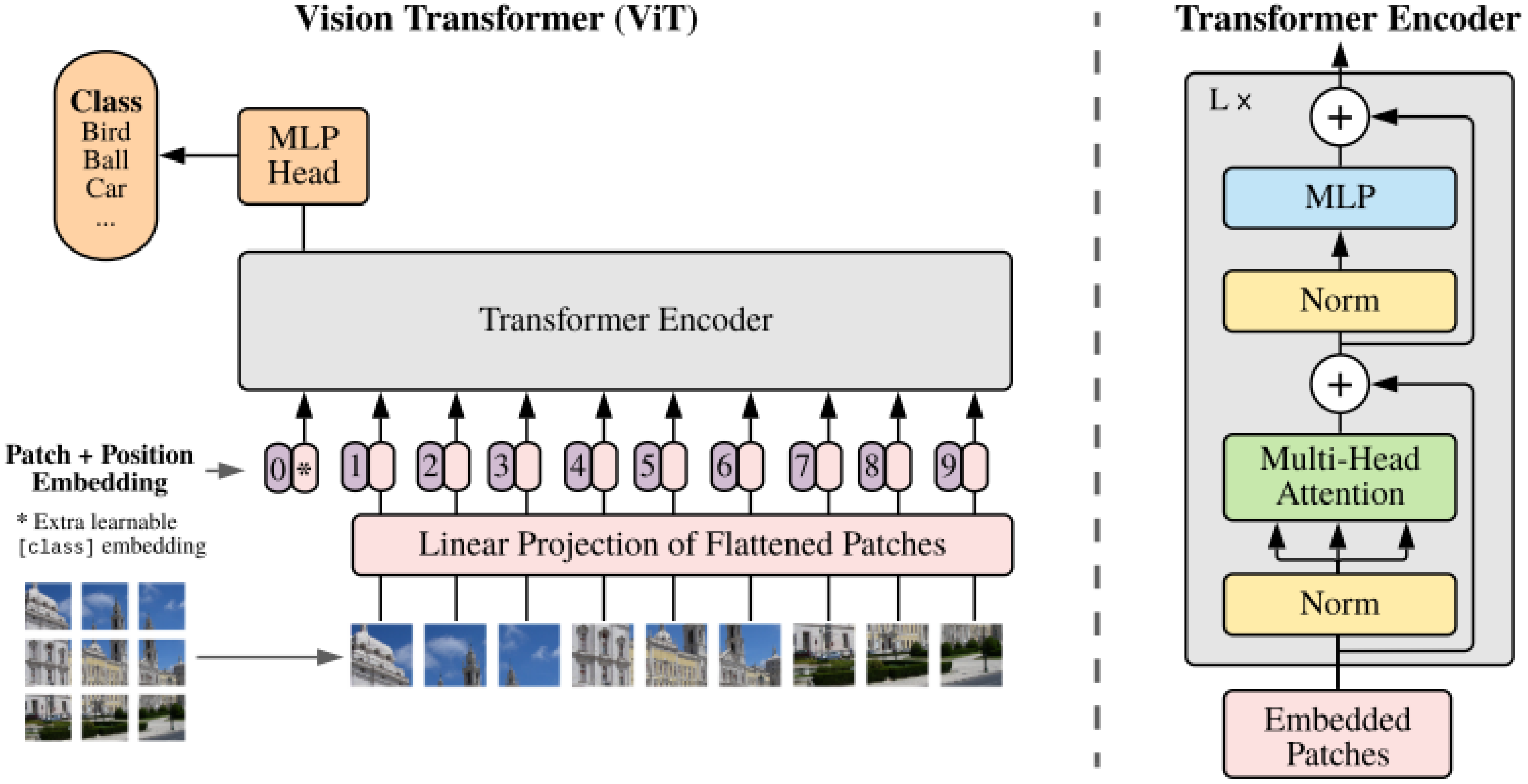


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

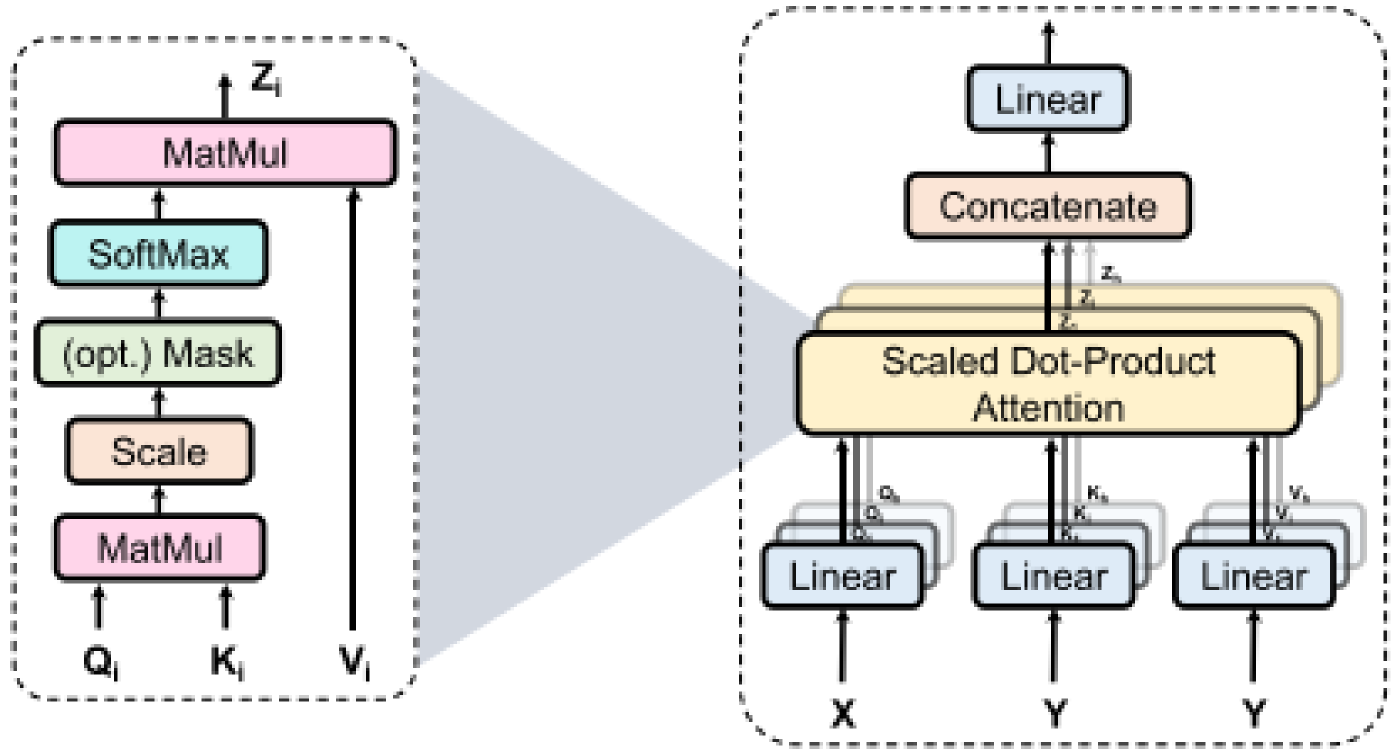


Fig. 2. The structure of the attention layer. Left: Scaled Dot-Product Attention. Right: Multi-Head Attention Mechanism.

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

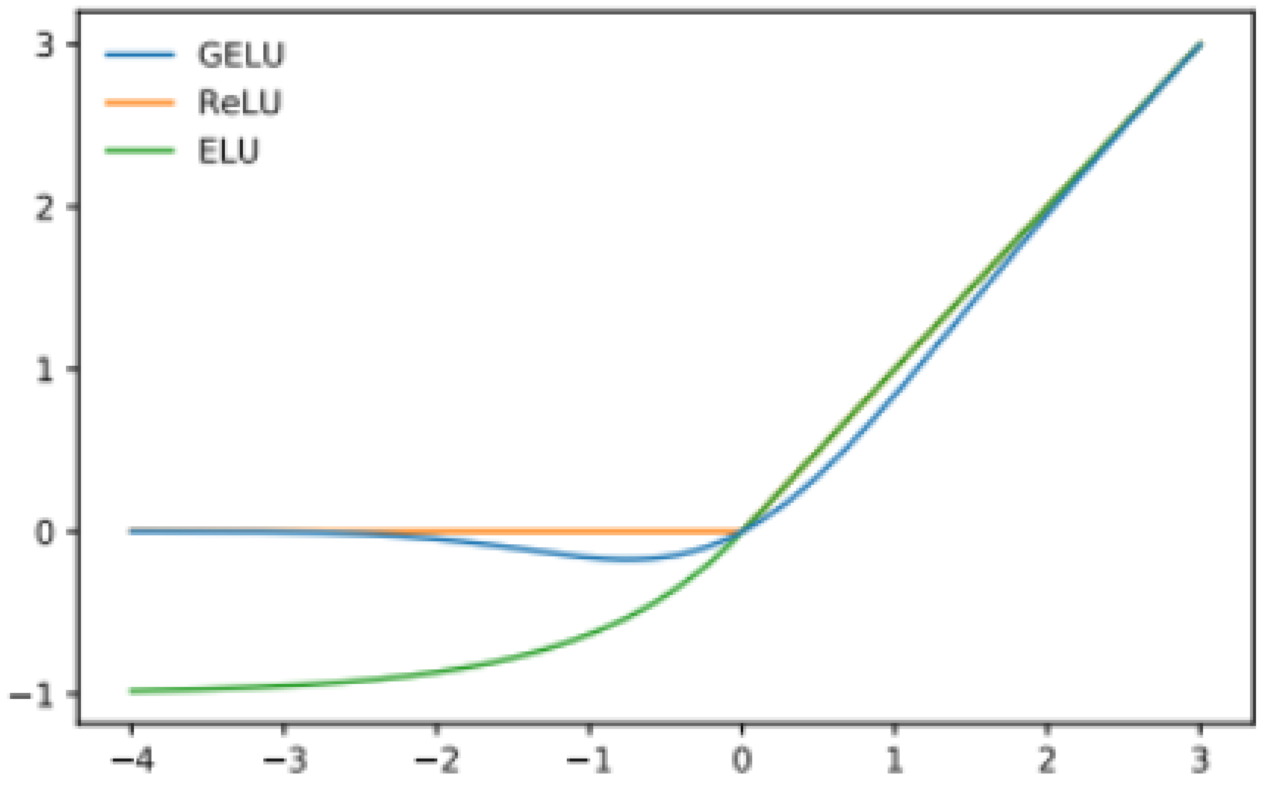


Figure1: A. Dosovitskiy, et al. An image is worth 16x16 words: Transformers for image recognition at scale.
Fig2: S. H. Khan, et al. Transformers in vision: A survey.

Post-Training Quantization for Vision Transformer

MEGVII 旷视

- **Motivation**
  - transformer has achieved remarkable performance on a variety of computer vision applications
  - vision transformers are often of sophisticated architectures, which are more difficult to be developed on mobile devices compared with CNN

- **Notation**

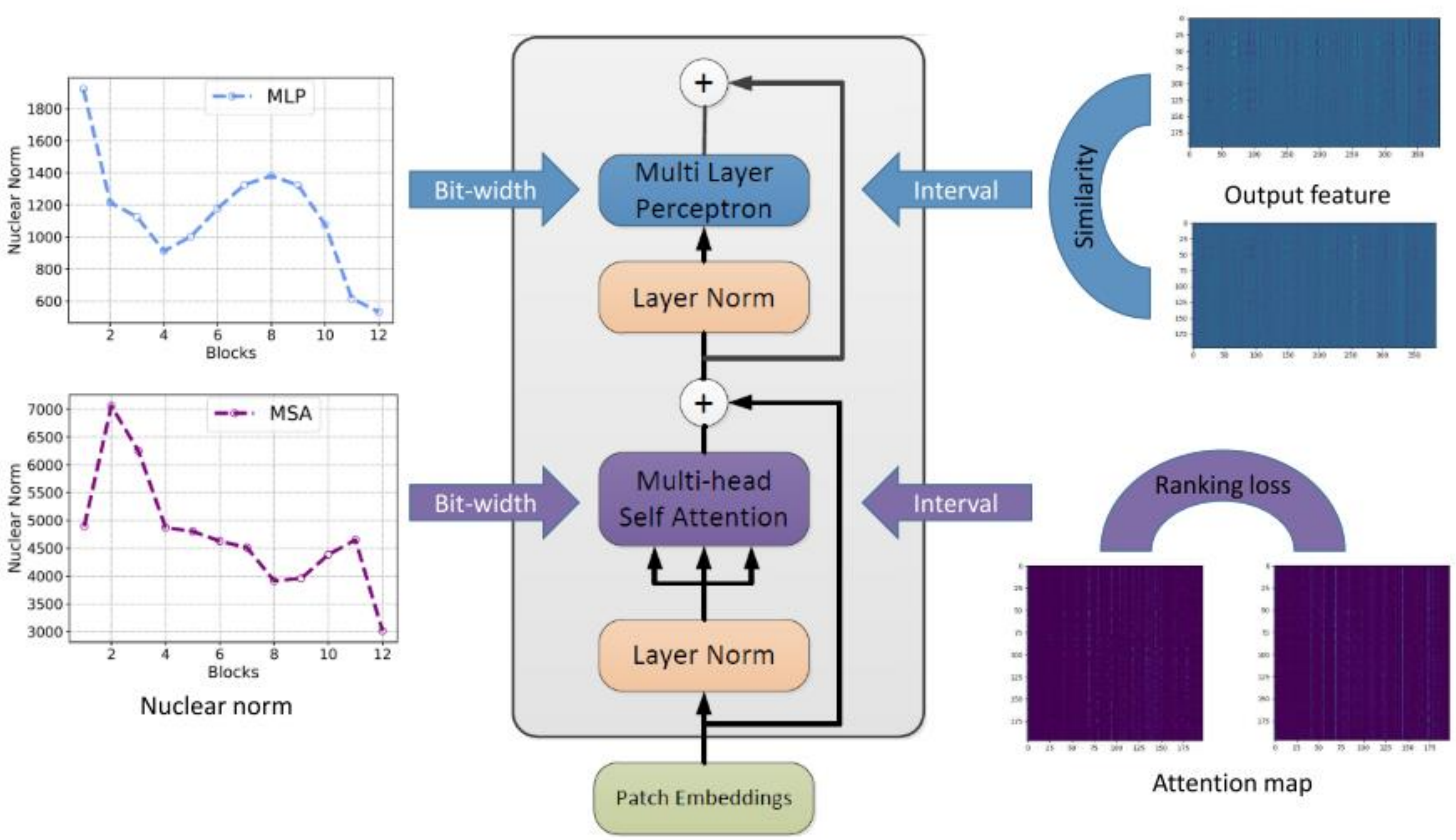
$$\Psi(X) = clip\left(round\left(\frac{X}{\Delta}\right), -2^{b-1}, 2^{b-1} - 1\right)$$

$$O_l = X_l W_l$$

$$\hat{O}_l = \Psi(X_l)\Psi(W_l)\Delta_l^X \Delta_l^W$$

Post-Training Quantization for Vision Transformer

- **Overview**



$$MLP(Z_l) = GeLU(Z_l W^1 + b^1)W^2 + b^2$$

$$MSA(X_l) = softmax(\frac{1}{\sqrt{d}} A_l) X_l W_l^V W_l^O$$

$$A_l = Q_l K_l^T = X_l W_l^Q W_l^{K^T} X_l^T$$

$$X_1 = \left[x_{class}; I_1^p W^E; \dots; I_1^p W^E\right] + E^{pos},$$

where $W^E \in \mathbb{R}^{(p^2 C)\times d}, E^{pos} \in \mathbb{R}^{(n+1)\times d}$

$$\Gamma(\hat{O}, O) = \frac{\sum_{j=1}^{m}(O_j - \bar{O})(\widehat{O_j} - \bar{\hat{O}})}{\sqrt{\sum_{j=1}^{m}(O_j - \bar{O})^2} \sqrt{\sum_{j=1}^{m}(\widehat{O_j} - \bar{\hat{O}})^2}}$$

$$\mathcal{L}_{ranking} = \sum_{k=1}^{h} \sum_{i=1}^{w-1} \sum_{j=i+1}^{w} \phi((\widehat{A_{ki}} - \widehat{A_{kj}}) \cdot sign(A_{ki} - A_{kj}))$$

Post-Training Quantization for Vision Transformer

MEGVII 旷视

- **Loss Function**

$$\max_{\triangle_l^W \triangle_l^X} \frac{1}{N}\sum_{i=1}^{N} \Gamma(O_l^i, \hat{O}_l^i) - \gamma \mathcal{L}_{ranking} \quad s.t.\ \ \triangle_l^W, \triangle_l^X \in \mathbb{R}^+$$

- **Experimental Results**

DeiT-S	ImageNet	Baseline	32	32	88	79.8
		Percentile [18]	6	6	16.5	70.49
		EasyQuant [30]	6	6	16.5	73.26
		Bit-Split [28]	6	6	16.5	74.04
		Ours	6	6	16.5	**74.58**
		Ours	6 MP	6 MP	16.6	**75.10**
		Percentile [18]	8	8	22.0	73.98
		EasyQuant [30]	8	8	22.0	76.59
		Bit-Split [28]	8	8	22.0	77.06
		Ours	8	8	22.0	**77.47**
		Ours	8 MP	8 MP	22.2	**78.09**
DeiT-B	ImageNet	Baseline	32	32	344	81.8
		Percentile [18]	6	6	64.5	73.99
		EasyQuant [30]	6	6	64.5	75.86
		Bit-Split [28]	6	6	64.5	76.39
		Ours	4 MP	4 MP	43.6	75.94
		Ours	6	6	64.5	**77.02**
		Ours	6 MP	6 MP	64.3	**77.47**
		Percentile [18]	8	8	86.0	75.21
		EasyQuant [30]	8	8	86.0	79.36
		Bit-Split [28]	8	8	86.0	79.42
		Ours	8	8	86.0	**80.48**
		Ours	8 MP	8 MP	86.8	**81.29**

FQ-ViT: Post-Training Quantization for Fully Quantized Vision Transformer

Yang Lin[*†], **Tianyu Zhang**[*], **Peiqin Sun**[‡], **Zheng Li**, **Shuchang Zhou**
MEGVII Technology
linyang.zhh@gmail.com, {zhangtianyu, sunpeiqin, lizheng02, zsc}@megvii.com

欢迎Star

https://github.com/megvii-research/FQ-ViT

Post-Training Quantization for Fully Quantized Vision Transformer

MEGVII 旷视

- **Motivation**
  - most existing quantization methods have been developed mainly on CNNs, and suffer severe degradation when applied to vision transformers.

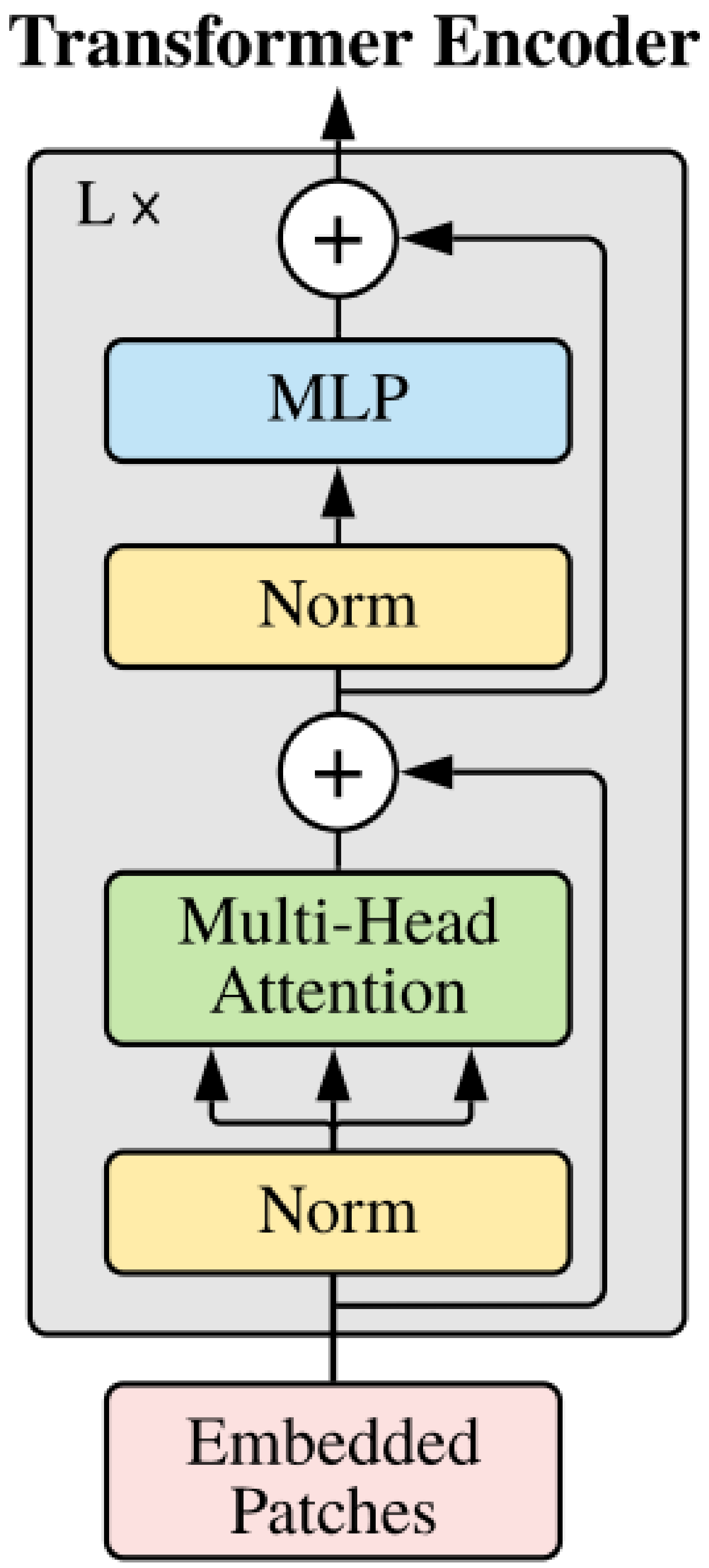
Method	W/A/Attn	DeiT-T	DeiT-S	DeiT-B	ViT-B	ViT-L	Swin-T	Swin-S	Swin-B
Full Precision	32/32/32	72.21	79.85	81.85	84.53	85.81	81.35	83.20	83.60
MinMax	8/8/8	70.94	75.05	78.02	23.64	3.37	64.38	74.37	25.58
EMA [Jacob *et al.*, 2018]	8/8/8	71.17	75.71	78.82	30.30	3.53	70.81	75.05	28.00
Percentile [Li *et al.*, 2019]	8/8/8	71.47	76.57	78.37	46.69	5.85	78.78	78.12	40.93
OMSE [Choukroun *et al.*, 2019]	8/8/8	71.30	75.03	79.57	73.39	11.32	79.30	78.96	48.55

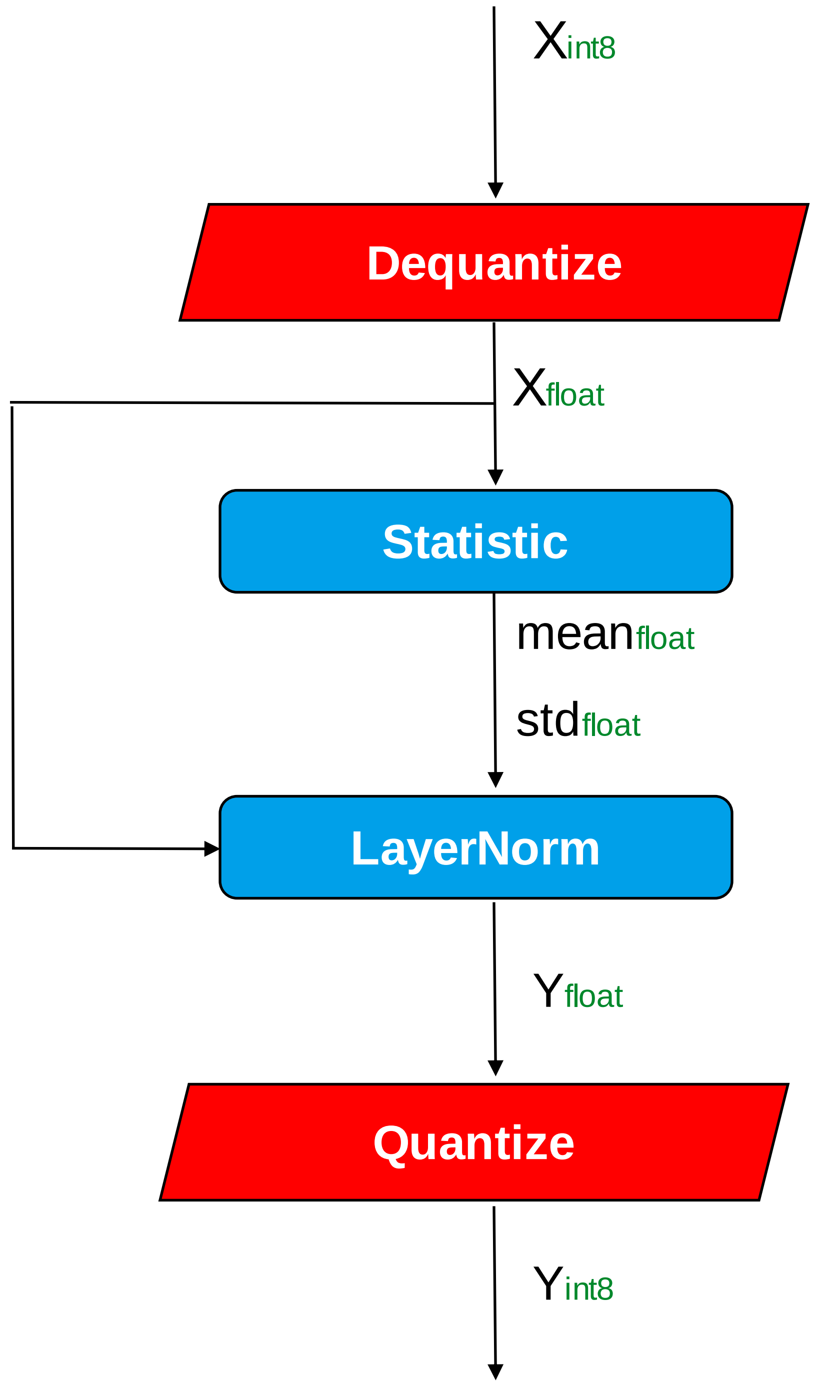
Method	W/A/Attn	Mask R-CNN w/ Swin-S	Cascade Mask R-CNN w/ Swin-S
Full Precision	32/32/32	48.5	52.0
MinMax	8/8/8	32.8	35.2
EMA [Jacob *et al.*, 2018]	8/8/8	37.9	40.4
Percentile [Li *et al.*, 2019]	8/8/8	41.6	44.7
OMSE [Choukroun *et al.*, 2019]	8/8/8	42.6	44.9

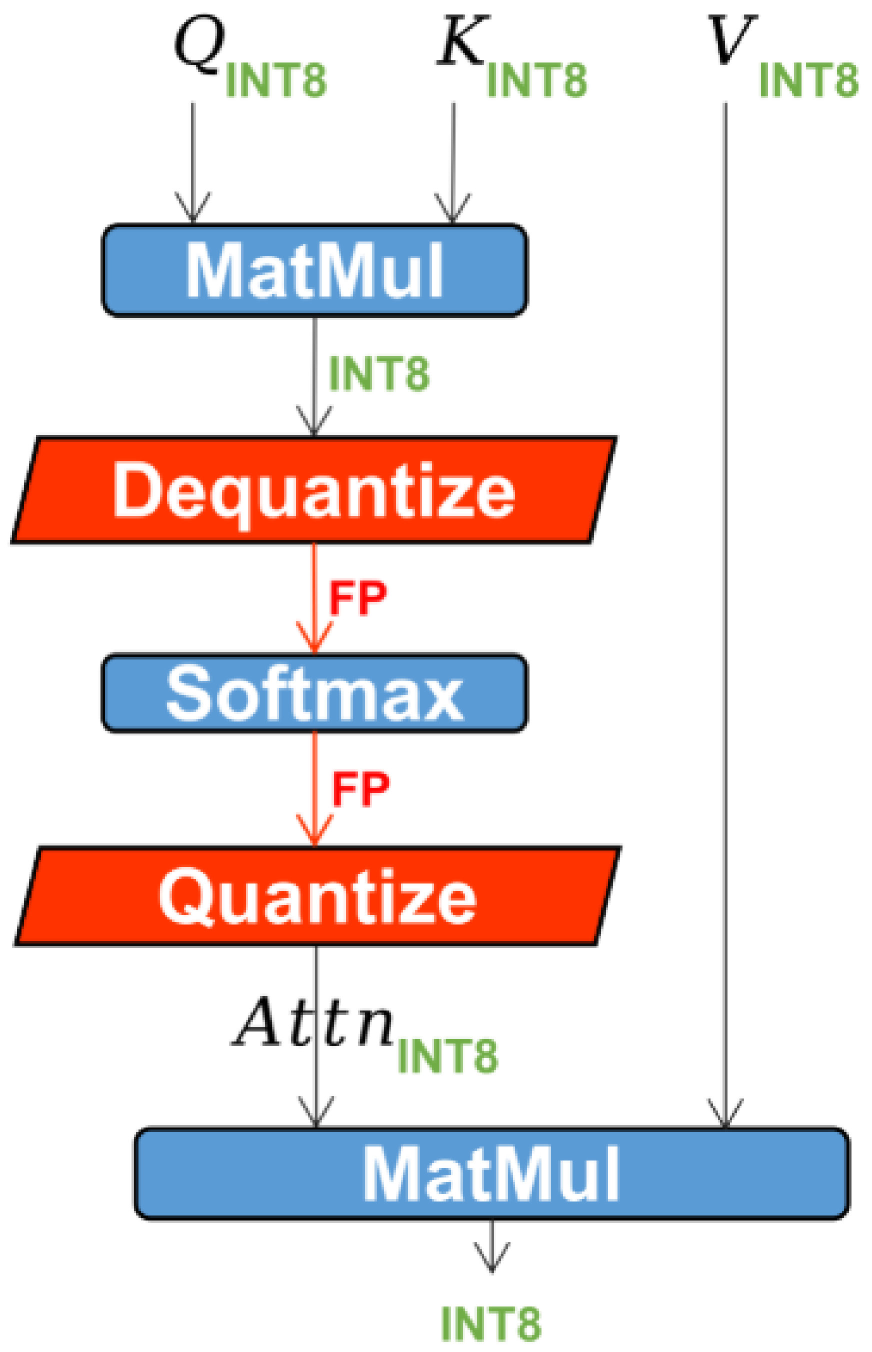
Post-Training Quantization for Fully Quantized Vision Transformer

MEGVII 旷视

- **Motivation**
  - current methods only quant matmul which cause the frequently quant and dequant in inference







MSA w/ Full Precision Softmax

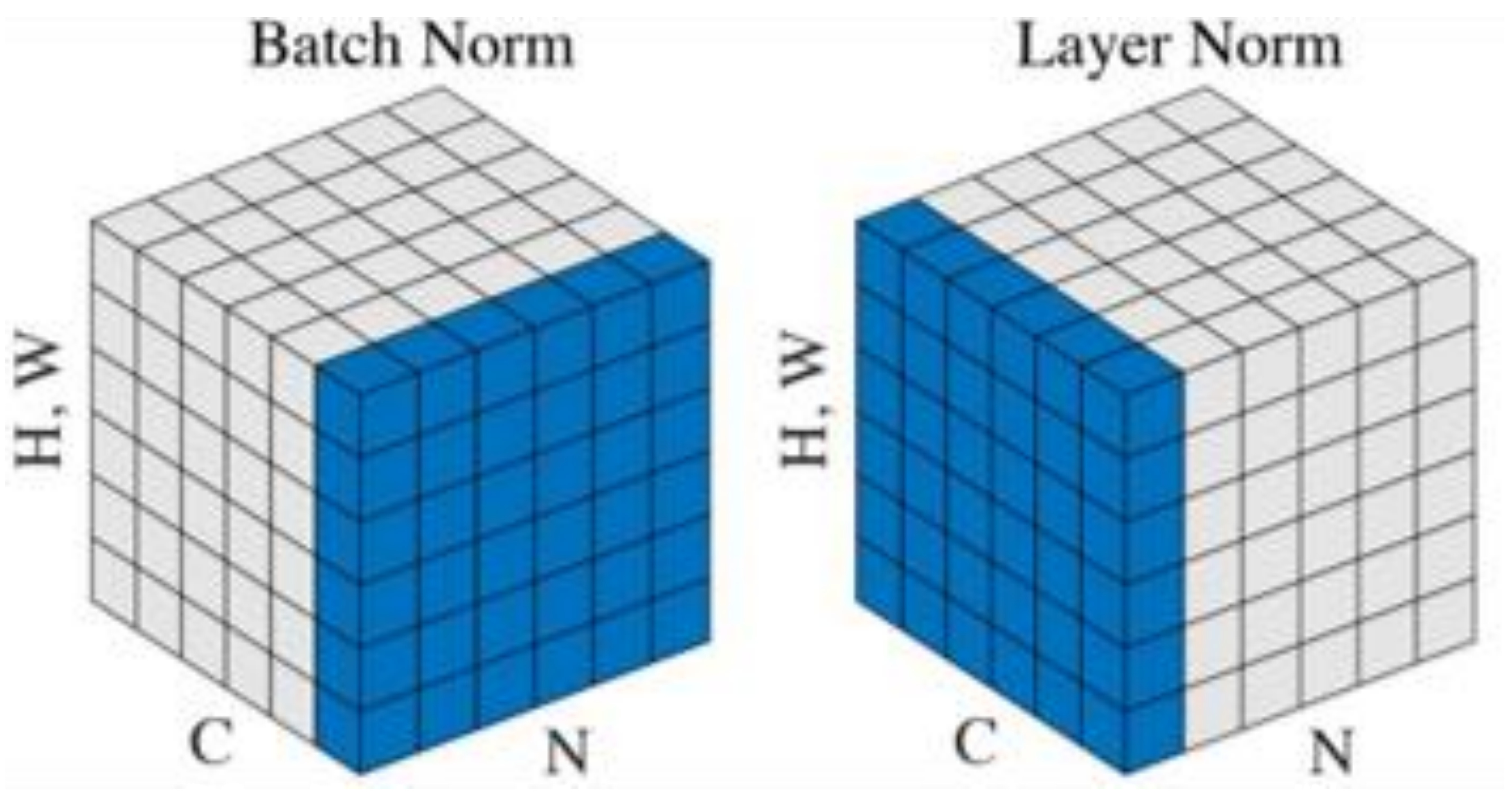
Post-Training Quantization for Fully Quantized Vision Transformer

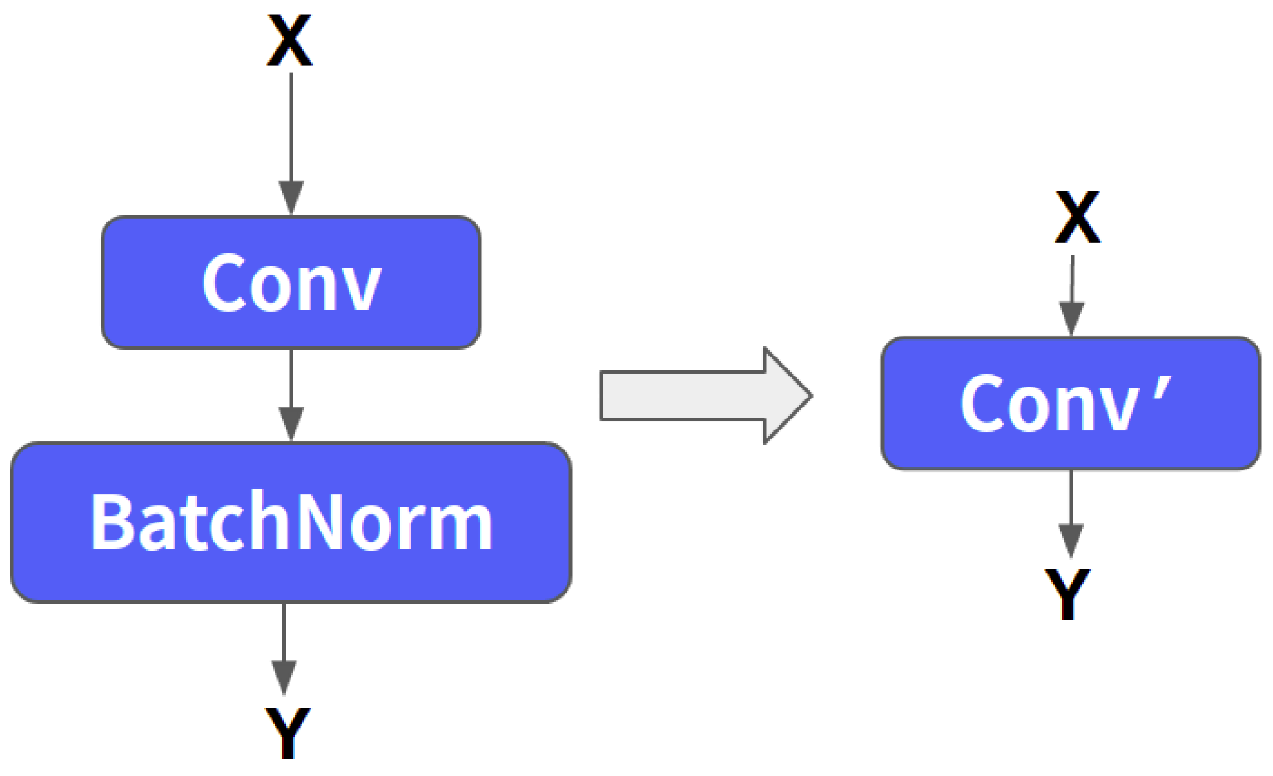
MEGVII 旷视

- BatchNorm v.s. LayerNorm

$$BatchNorm(X) = \frac{X - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} * \gamma + \beta$$

$$LayerNorm(X) = \frac{X - \mu_X}{\sqrt{\sigma_X^2 + \epsilon}} * \gamma + \beta$$





- BN stats can be computed offline, and absorbed when inference
- LN stats are computed online, must be computed separately.

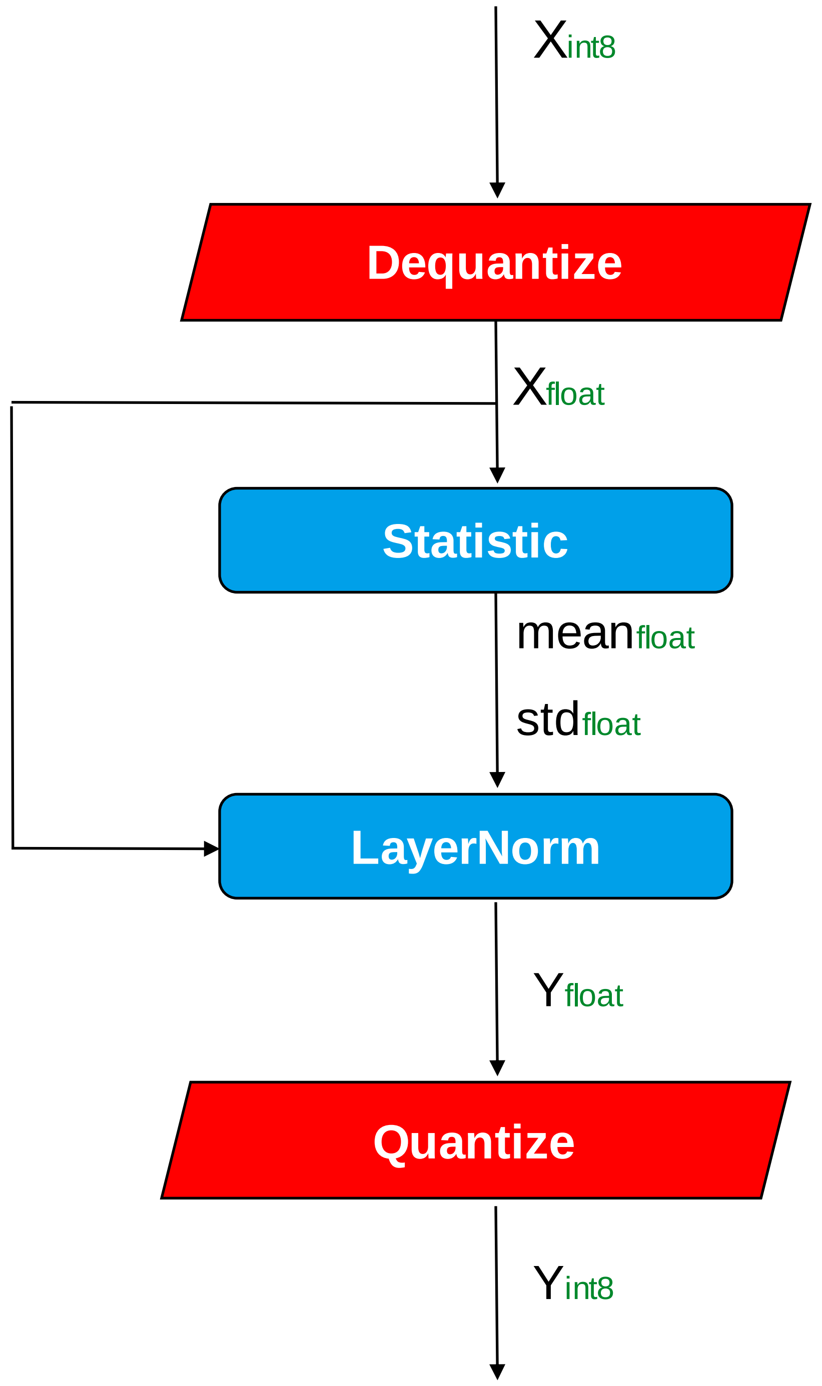
Post-Training Quantization for Fully Quantized Vision Transformer

MEGVII 旷视

- Power-of-Two approximation for Scaling Factor of LayerNorm Quantization

Res50: 21.6 / 4.2

Swin-B: 622.5/15.5



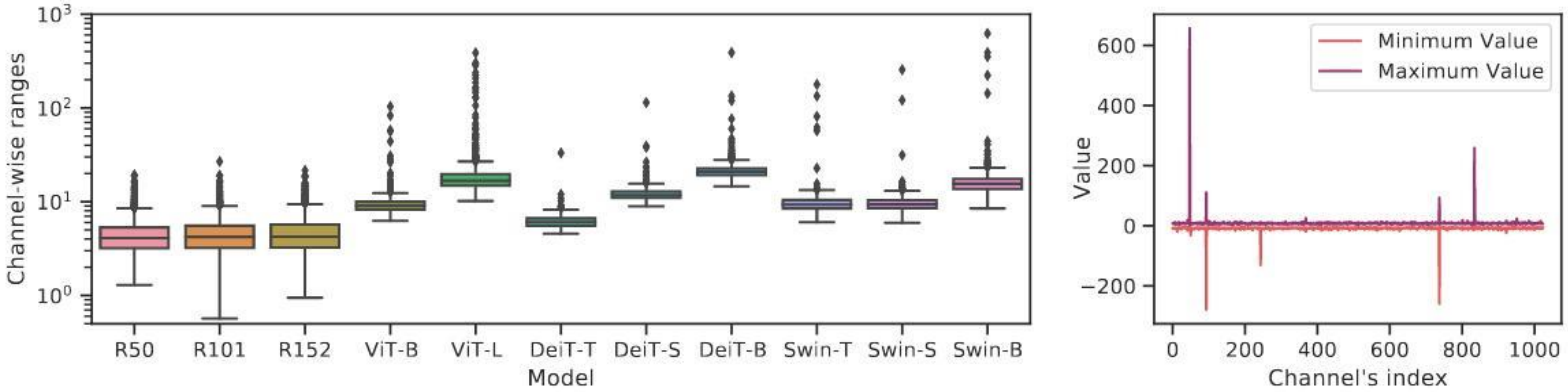


Figure 2: **Left**: Boxplot of the last LayerNorm inputs' channel-wise ranges in each model. **Right**: Channel-wise minimum and maximum values of the last LayerNorm inputs in full precision Swin-B. The above two figures show that there exists more serious inter-channel variation in vision transformers than CNNs, which leads to unacceptable quantization errors with layer-wise quantization.

$$\widetilde{x_q^d} = x_q^d \ll \alpha^d$$

$$\mu(X) \approx s * \mu(\widetilde{X_q}) = s * \frac{1}{D} \sum_{d=1}^{D} \widetilde{x_q^d}$$

$$\sigma(X) \approx s * \sigma(\widetilde{X_q}) = s * \frac{1}{D} \sum_{d=1}^{D} [\widetilde{x_q^d} - \mu(\widetilde{X_q})]^2$$

- Log-Int-Softmax for Softmax Quantization

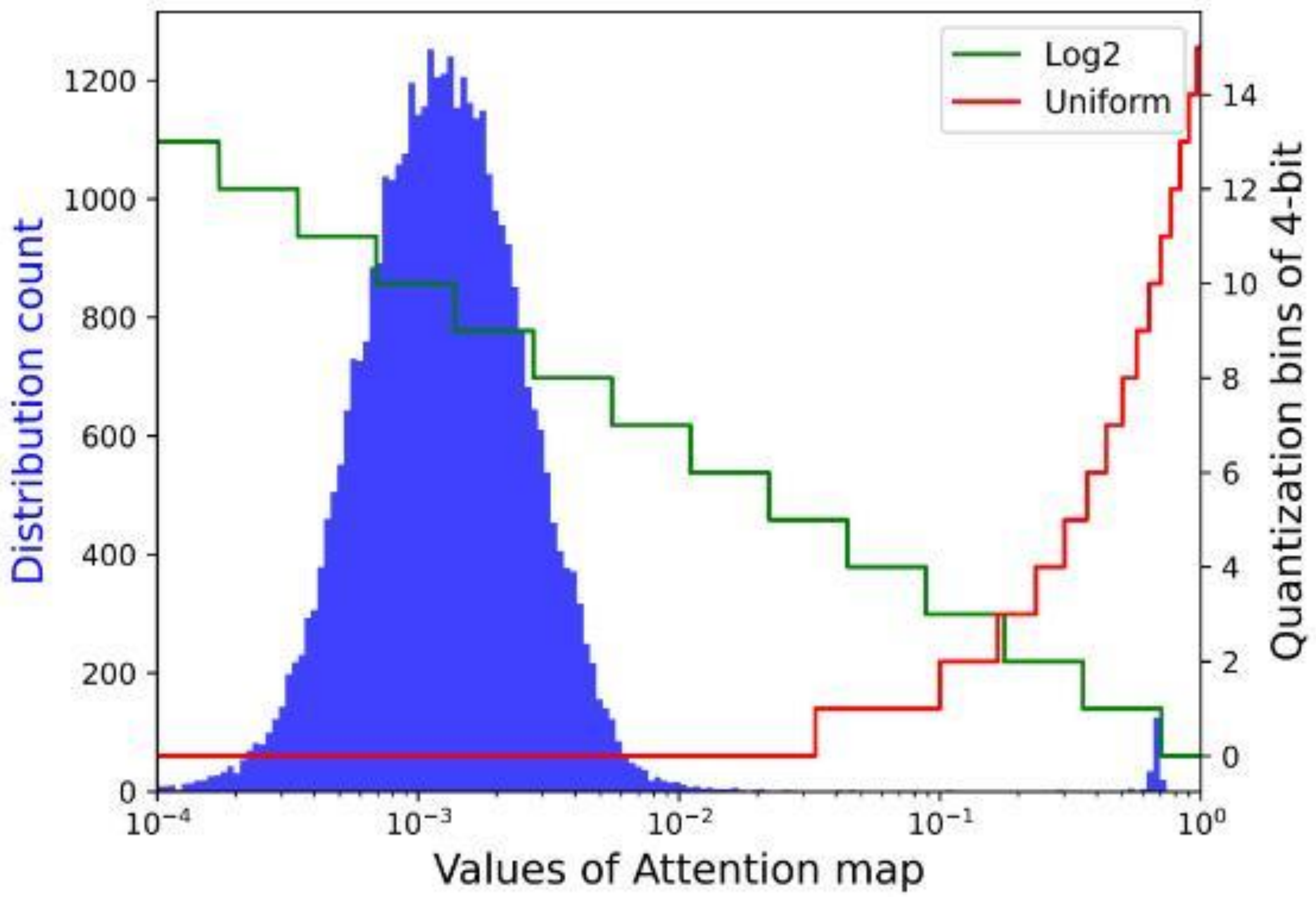


Figure 3: Distribution of attention maps in ViT-L with visualizing the 4-bit quantized bins of uniform and log2 quantization. X-axis is in log-scale and we can observe that log2 quantization preserves more bins than uniform for small values.

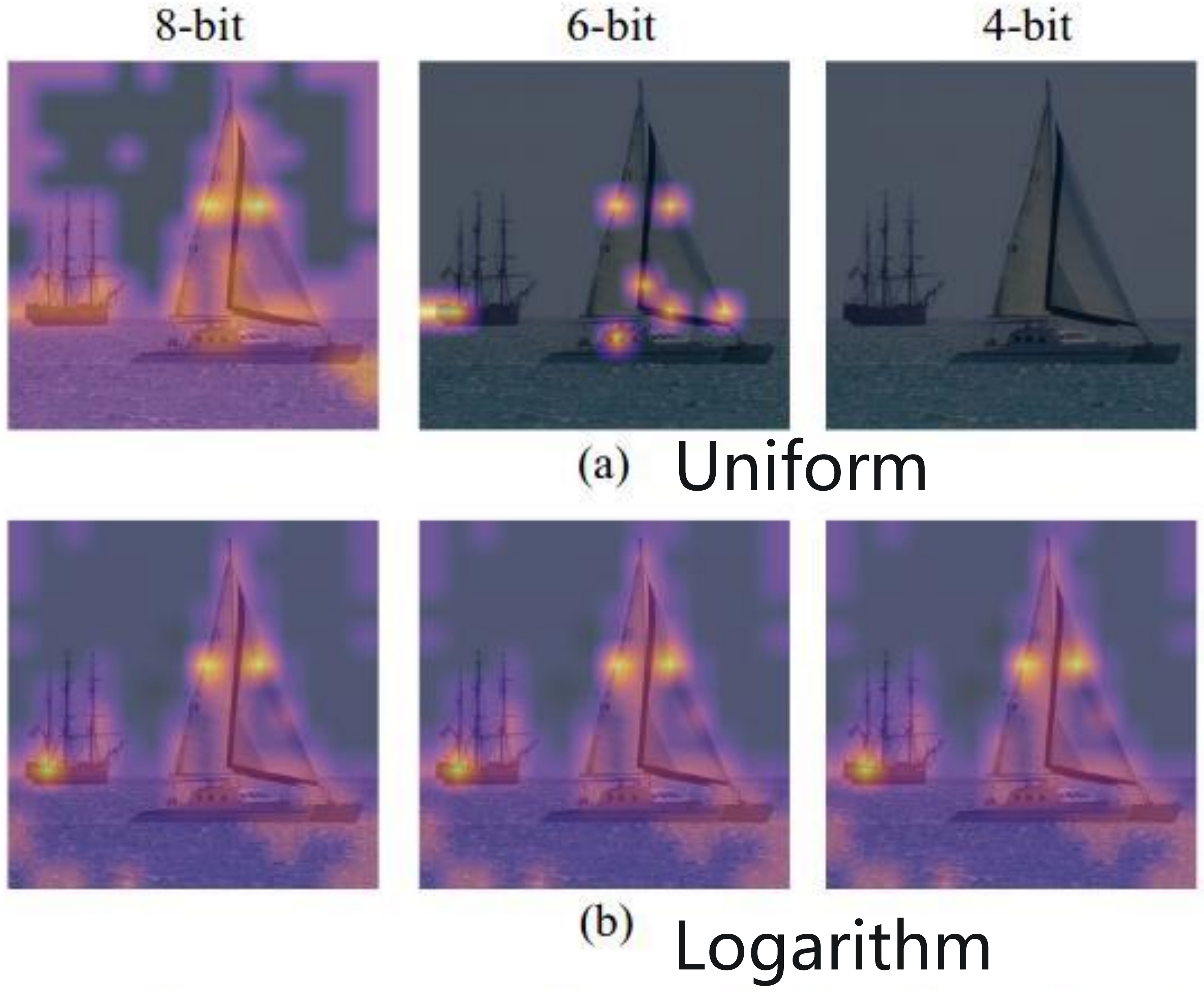


Figure 5: Attention map visualization. (a) shows the results of uniform quantization and (b) shows the results of our Log-Int-Softmax.

Post-Training Quantization for Fully Quantized Vision Transformer

MEGVII 旷视

- Log-Int-Softmax for Softmax Quantization

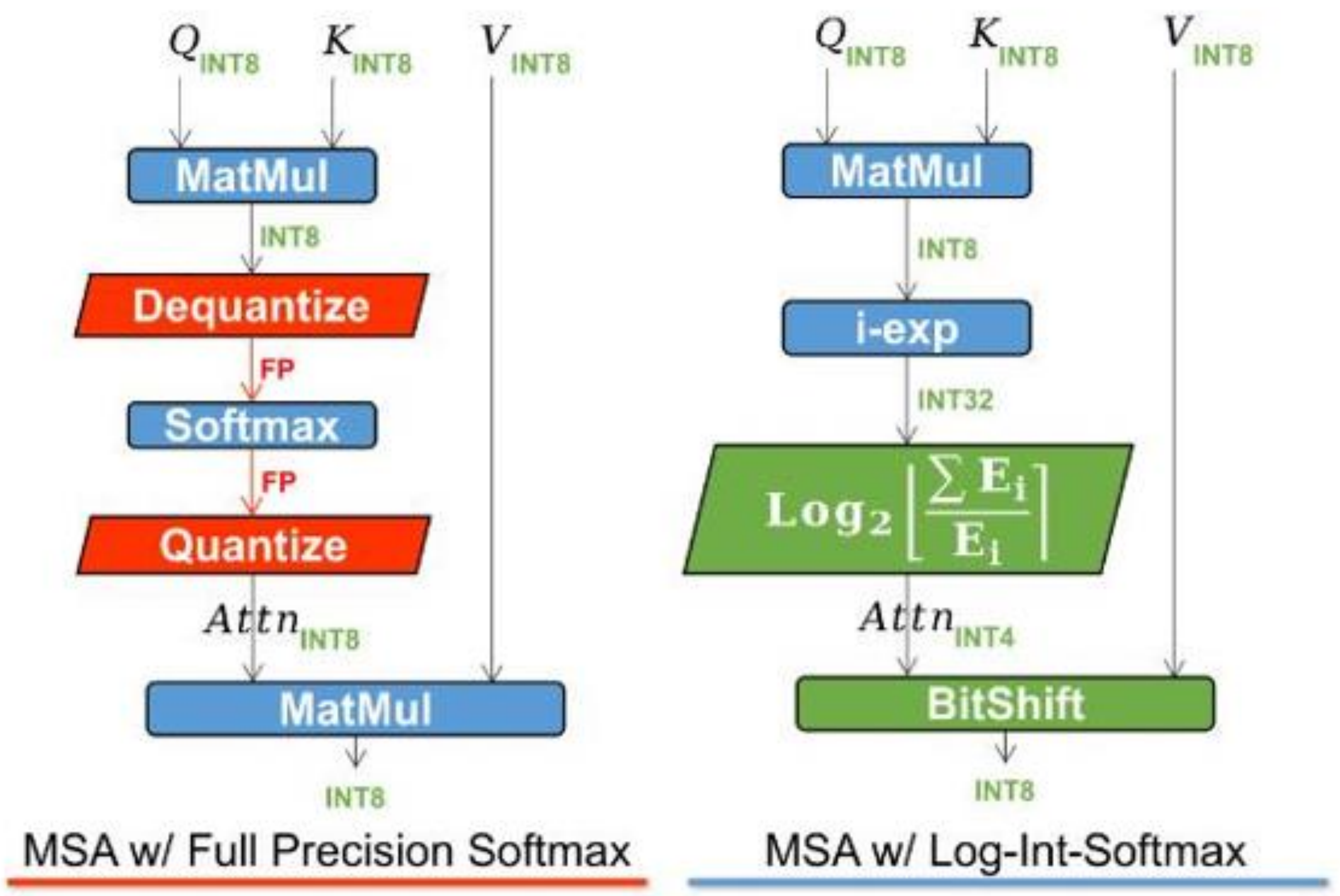


Figure 4: Comparison of using full precision Softmax and Log-Int-Softmax in quantized multi-head self-attention inference. Full precision Softmax needs to dequantize and requantize around Softmax, while LIS keeps an integer-only data type in the whole MSA inference.

$$A = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

$$softmax(s * X_q) \approx \frac{i\_\exp(x_q^i, s)}{\sum_{d=1}^{D} i\_\exp(x_q^d, s)}$$

$$A_q = clip\big(round\big(-log2(A)\big), 0, 2^b - 1\big)$$

$$A \cdot V_q = 2^{-A_q} \cdot V_q = V_q \gg A_q = \frac{1}{2^N}\big(V_q \ll (N - A_q)\big)$$

S. Kim, A. Gholami, et al. I-BERT: Integer-only BERT Quantization

Post-Training Quantization for Fully Quantized Vision Transformer

MEGVII 旷视

- The first work to implement fully quantized vision transformer and achieves a nearly lossless quantization.
- Even with an aggressively low-bit (4-bit) on attention maps, FQ-ViT also has encouraging results.
- FQ-ViT can generalize well to different tasks.

Method	W/A/Attn	DeiT-T	DeiT-S	DeiT-B	ViT-B	ViT-L	Swin-T	Swin-S	Swin-B
Full Precision	32/32/32	72.21	79.85	81.85	84.53	85.81	81.35	83.20	83.60
MinMax	8/8/8	70.94	75.05	78.02	23.64	3.37	64.38	74.37	25.58
EMA [Jacob *et al.*, 2018]	8/8/8	71.17	75.71	78.82	30.30	3.53	70.81	75.05	28.00
Percentile [Li *et al.*, 2019]	8/8/8	71.47	76.57	78.37	46.69	5.85	78.78	78.12	40.93
OMSE [Choukroun *et al.*, 2019]	8/8/8	71.30	75.03	79.57	73.39	11.32	79.30	78.96	48.55
Bit-Split* [Wang *et al.*, 2020]	8/8/8	-	77.06	79.42	-	-	-	-	-
PTQ for ViT* [Liu *et al.*, 2021b]	8/8/8	-	77.47	80.48	-	-	-	-	-
FQ-ViT	8/8/8	**71.61**	**79.17**	**81.20**	**83.31**	**85.03**	**80.51**	**82.71**	**82.97**
	8/8/4	**71.07**	**78.40**	**80.85**	**82.68**	**84.89**	**80.04**	**82.47**	**82.38**

Table 1: Comparison of the top-1 accuracy with state-of-the-art methods on ImageNet dataset. * indicates that all LayerNorm and Softmax modules are not quantized.

Method	W/A/Attn	Mask R-CNN w/ Swin-S	Cascade Mask R-CNN w/ Swin-S
Full Precision	32/32/32	48.5	52.0
MinMax	8/8/8	32.8	35.2
EMA [Jacob *et al.*, 2018]	8/8/8	37.9	40.4
Percentile [Li *et al.*, 2019]	8/8/8	41.6	44.7
OMSE [Choukroun *et al.*, 2019]	8/8/8	42.6	44.9
FQ-ViT	8/8/8	**47.8**	**51.4**
	8/8/4	**47.2**	**50.8**

Table 2: Comparison of the bbox mAP with state-of-the-art methods on COCO dataset.

The Sparsebit framework

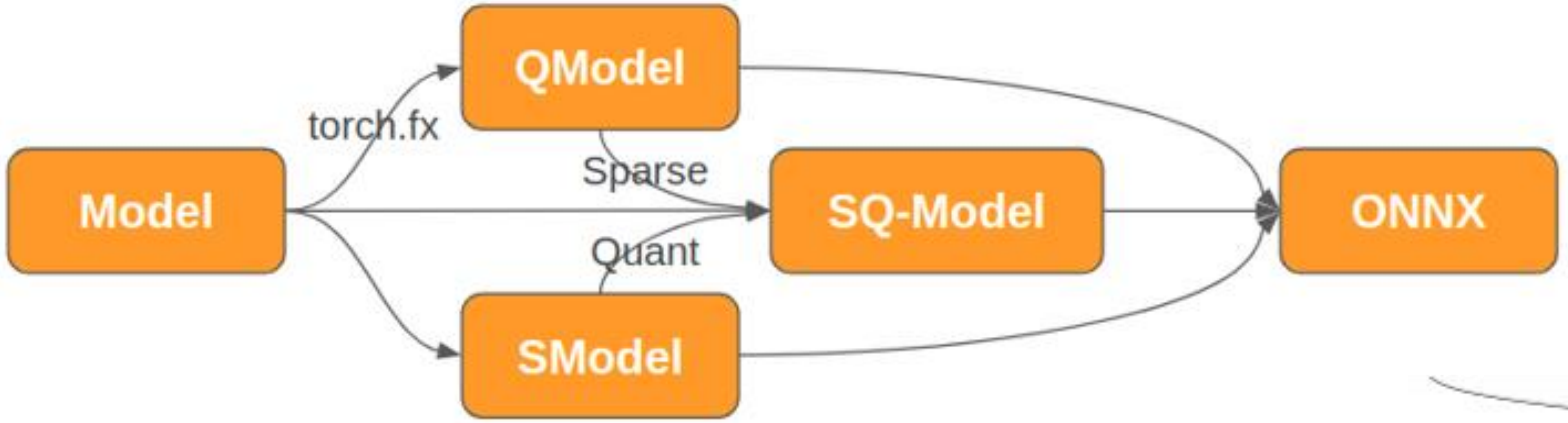
- **Introduction**
  - A toolbox includes quantization tools and pruning (sparse) tools for neural networks
  - Can be easily integrated into other projects as a plugin
  - Easy to extend QModule / Quantizer / Observer
  - QModel can be exported to a QDQ ONNX which TensorRT/ONNXRuntime can load directly
- **Schedules**

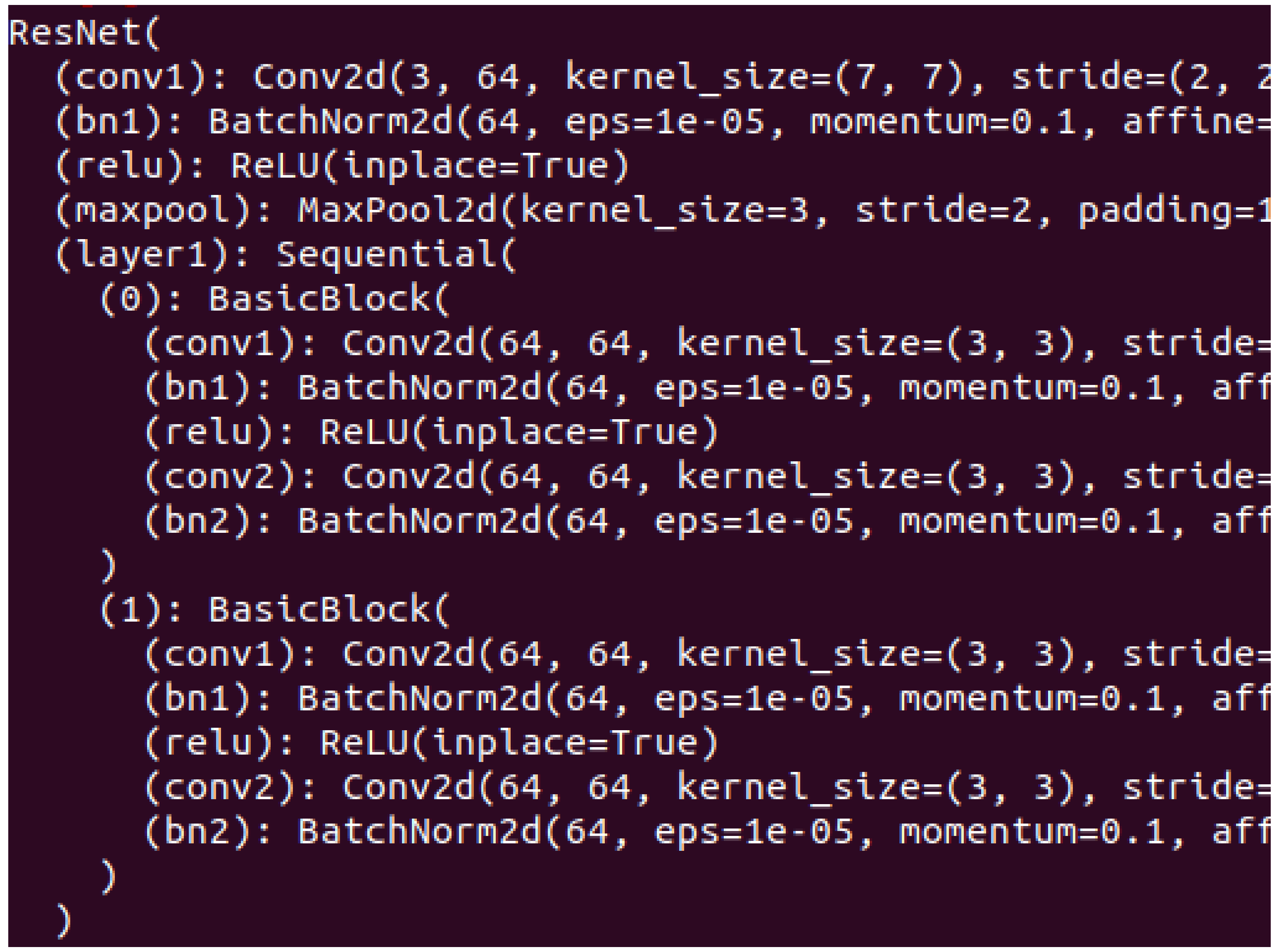
content	release time
release a quantization toolbox	08.12
release a pruning toolbox	08.26
update SSQL	09.01
a deployment demo of FQ-ViT	09.10

https://github.com/megvii-research/Sparsebit

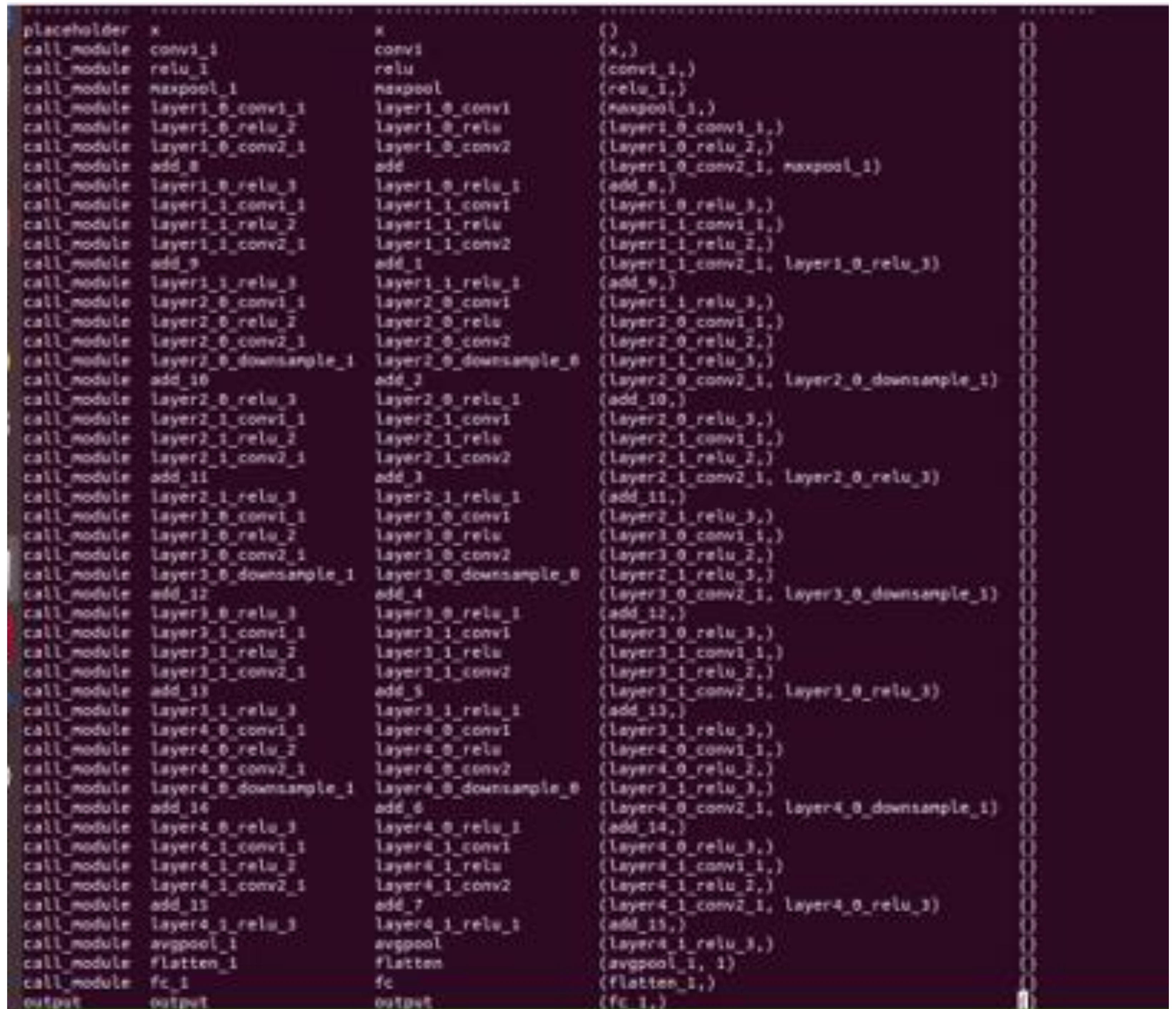
Sparsebit

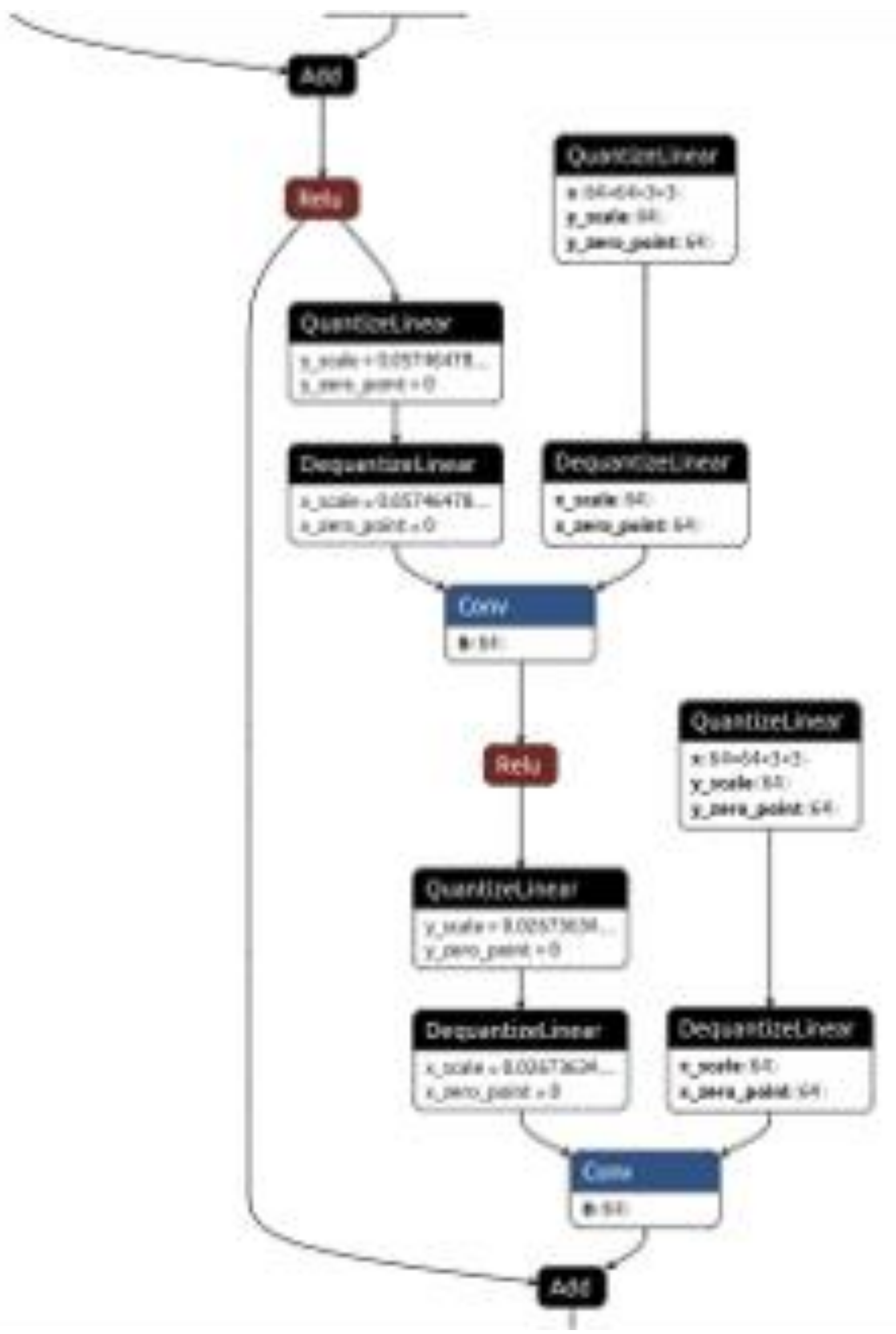
- ## Model Conversion Flow





```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, aff
    )
  )
```





Sparsebit

MEGVII 旷视

- **User Guide**
  - Convert Model to QModel

```
qconfig = parse_qconfig(args.qconfig)
qmodel = QuantModel(model, config=qconfig)
```

  - Calibration

```
qmodel.prepare_calibration()
calib_size, cur_size = 256, 0
qmodel.eval()
with torch.no_grad():
    for data, target in trainloader:
        qmodel(data.cuda())
        cur_size += data.shape[0]
        if cur_size >= calib_size:
            break
qmodel.calc_qparams()
```

  - Export to ONNX

```
qmodel.export_onnx(
    torch.randn(1, 3, 224, 224),
    input_names=["input"],
    output_names=["output"],
    name="qresnet18.onnx"
)
```

Sparsebit

- **Homeworks**
  - Q1: 请模仿cifar10样例, 构造imagenet工程并获取vgg16_bn / resnet18 / mobilenetv2的PTQ实验结果(8w8f).
  - Q2: 请基于resnet18实验, 把calibration-set里面的图片换成标准高斯噪声输入, 当calibration-set大小为1, 10, 100时, 请问精度分别是多少, 精度不是0或者很低的原因是什么呢?
  - Q3: 请增加moving-average observer，并重新运行题目一的Resnet18，观察实验结果。
  - Q4: 请使用trtexec分别测试导出的ONNX模型, 在batch=1, 32, 128, 256情况下, 相较于fp16的加速情况, 请分析 int8/fp16 为什么在batch不同时会有显著差异?
  - Q5: 仿造cifar10 QAT样例训练resnet18模型, 要求bit分别为4w4f, 2w4f, 观察其精度变化.

- Join Us
  - 小组常年招收实习生，欢迎有志于从事模型量化，模型稀疏与剪枝，模型蒸馏，自监督学习等方向的同学，可以投递简历至 sunpeiqin@megvii.com

Q & A