20:00准时开播

还没有课件PPT的同学，扫码添加 欣然助教 获取

- 领取课件PPT
- 参加交流群
- 获取论文

领取课件资料 截图扫码参加

大语言模型的量化推理加速技术基础

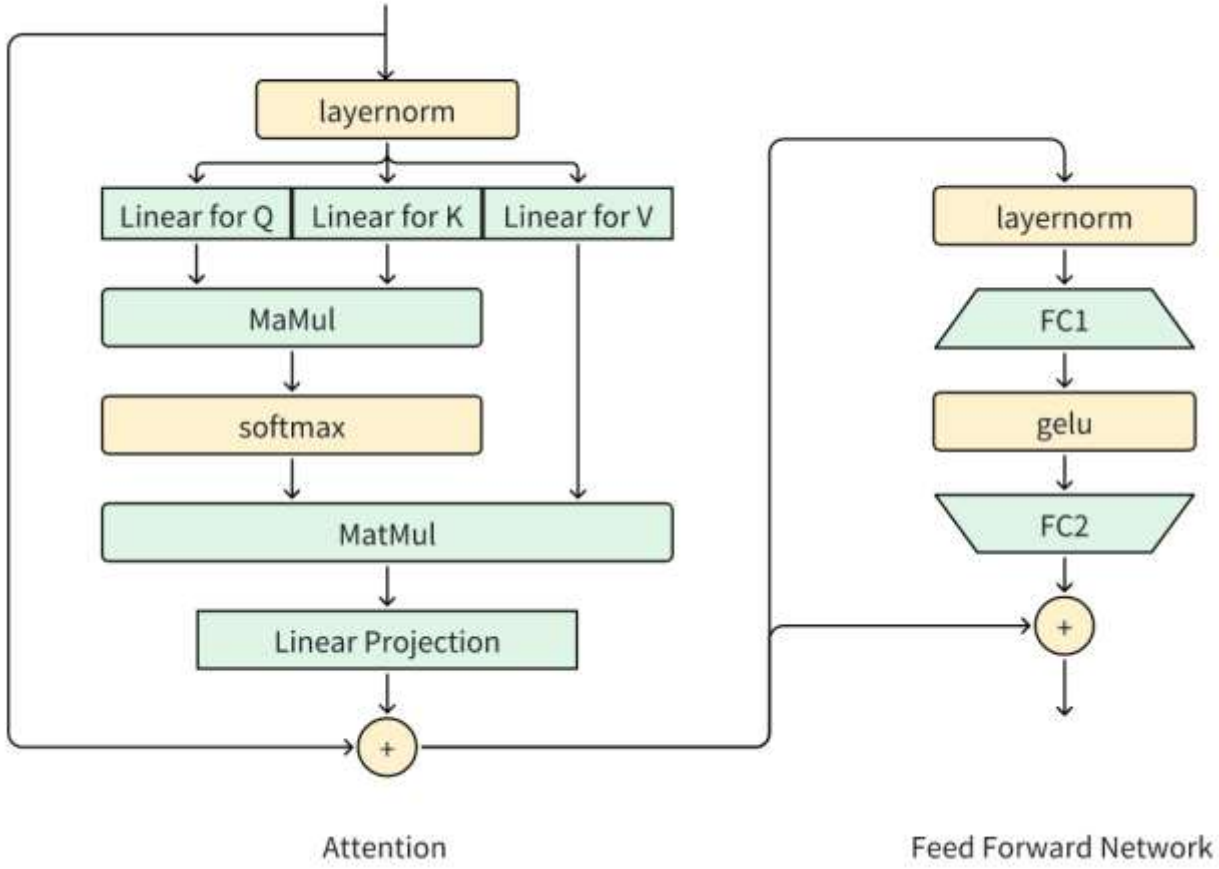
陈晓宇

2024.03.26

一、LLM 基础概念

LLM 模型结构

- LLM 的模型结构主要由 Transformer 中的 Decoder 部分堆叠而成，Decoder 层包括 **Attention** 和 **FFN** 两部分。
- LLM 模型的推理过程（平均 1 个 token 可以表示 1.5 个汉字）：



北京	有很多	好	玩的		

LLM Model

北京	有哪些	好	玩的	景点	?	北京	有很多	好	玩的		

- **Prefill 预填充阶段：**一次性接收 prompt 的输入，生成首字输出。
- **Decode 解码阶段：**自回归过程，当前输出依赖之前的输出，只能一个 token 一个 token 输出。

LLM 推理的关键指标

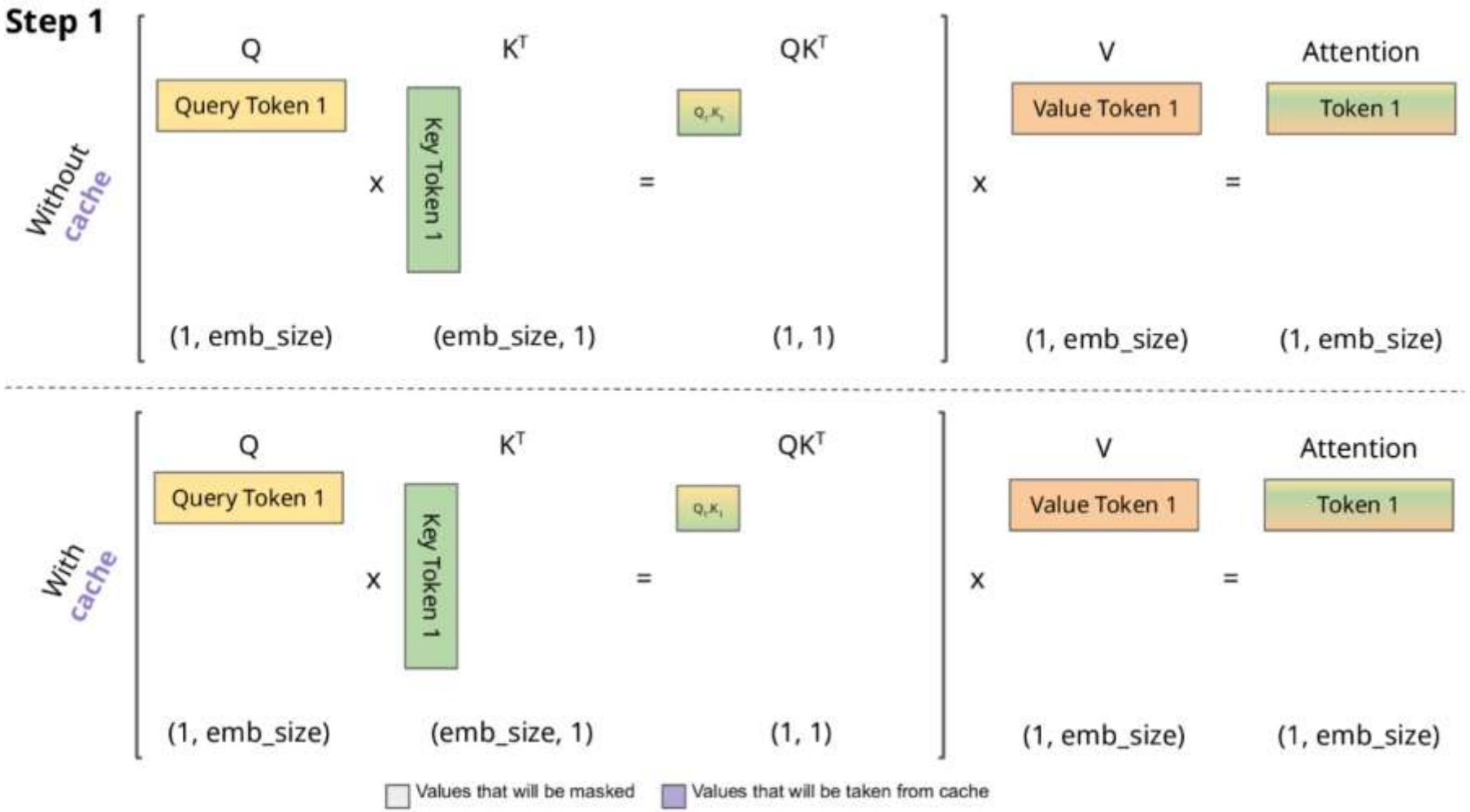
指标	说明	影响因素	优化方法
首字延迟 （TTFT，Time To First Token）	Prefill 阶段执行的时间。	batch_size、输入长度	优化计算、降低访存
解码单个输出的时间 （TPOT，Time Per Output Token）	Decode 阶段生成单个 token 的时间。	batch_size	降低访存
时延（Latency）	生成响应的总时间。时延 = TTFT + TPOT * token数。	batch_size、输入和输出长度	优化 prompt 设置或微调模型优化输出长度
吞吐量（Throughput）	服务器在所有用户请求中每秒可生成的 decoding token 数。	batch_size、输入长度	Continuous Batching

KV cache

计算 Q、K、V：$Q = XW^Q, K = XW^K, V = XW^V$

计算 Attention：$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

在模型推理时，由于是自回归模式，需要一个 token 一个 token 的计算，在计算当前 token 时，前面 token 的 QKV 在之前的步骤已经算过了，不需要再次计算，只需要将其保存提供给当前步的 Attention 计算使用，这一方法称为 KV Cache。

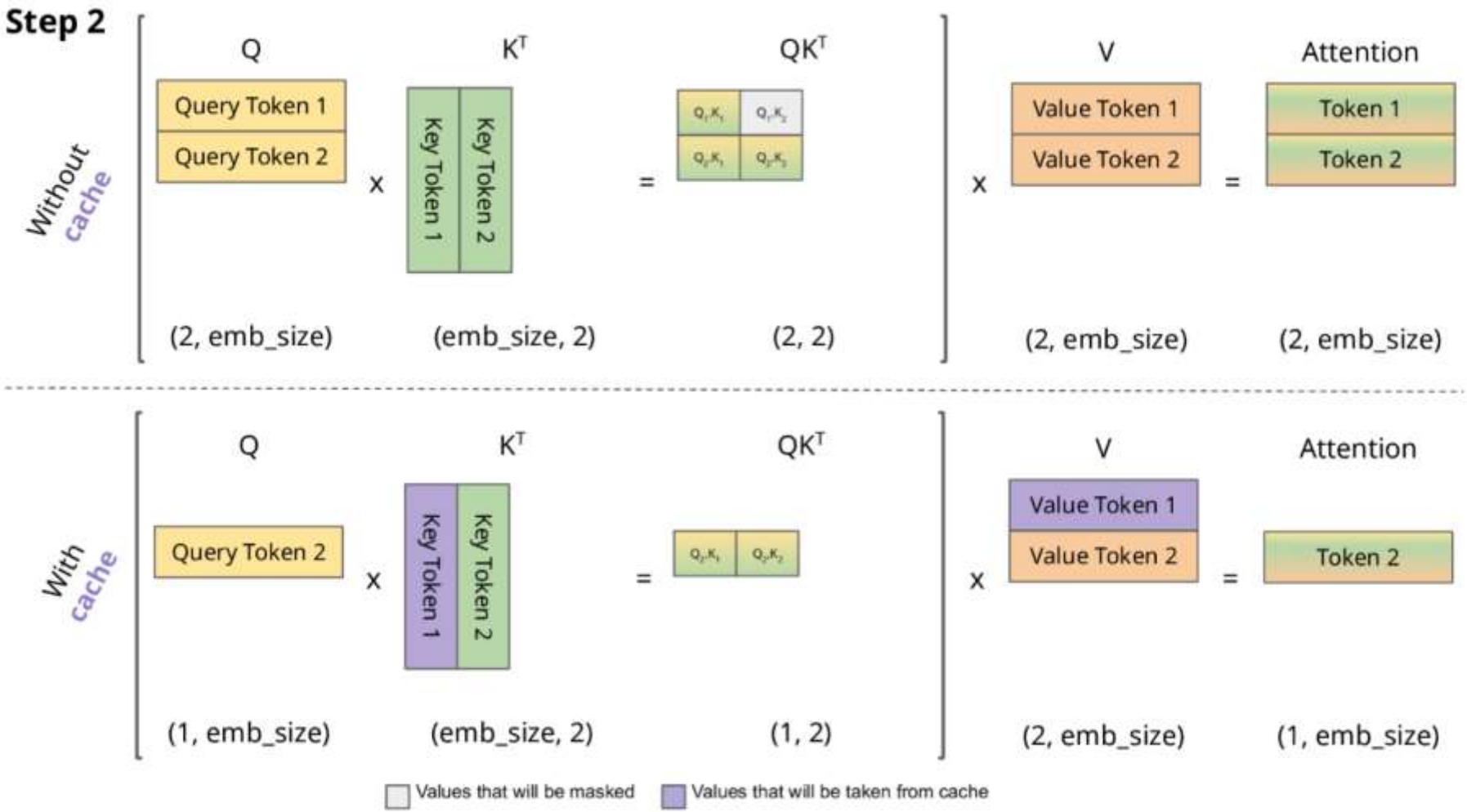


KV cache

计算 Q、K、V：$Q = XW^Q, K = XW^K, V = XW^V$

计算 Attention：$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

在模型推理时，由于是自回归模式，需要一个 token 一个 token 的计算，在计算当前 token 时，前面 token 的 QKV 在之前的步骤已经算过了，不需要再次计算，只需要将其保存提供给当前步的 Attention 计算使用，这一方法称为 KV Cache。

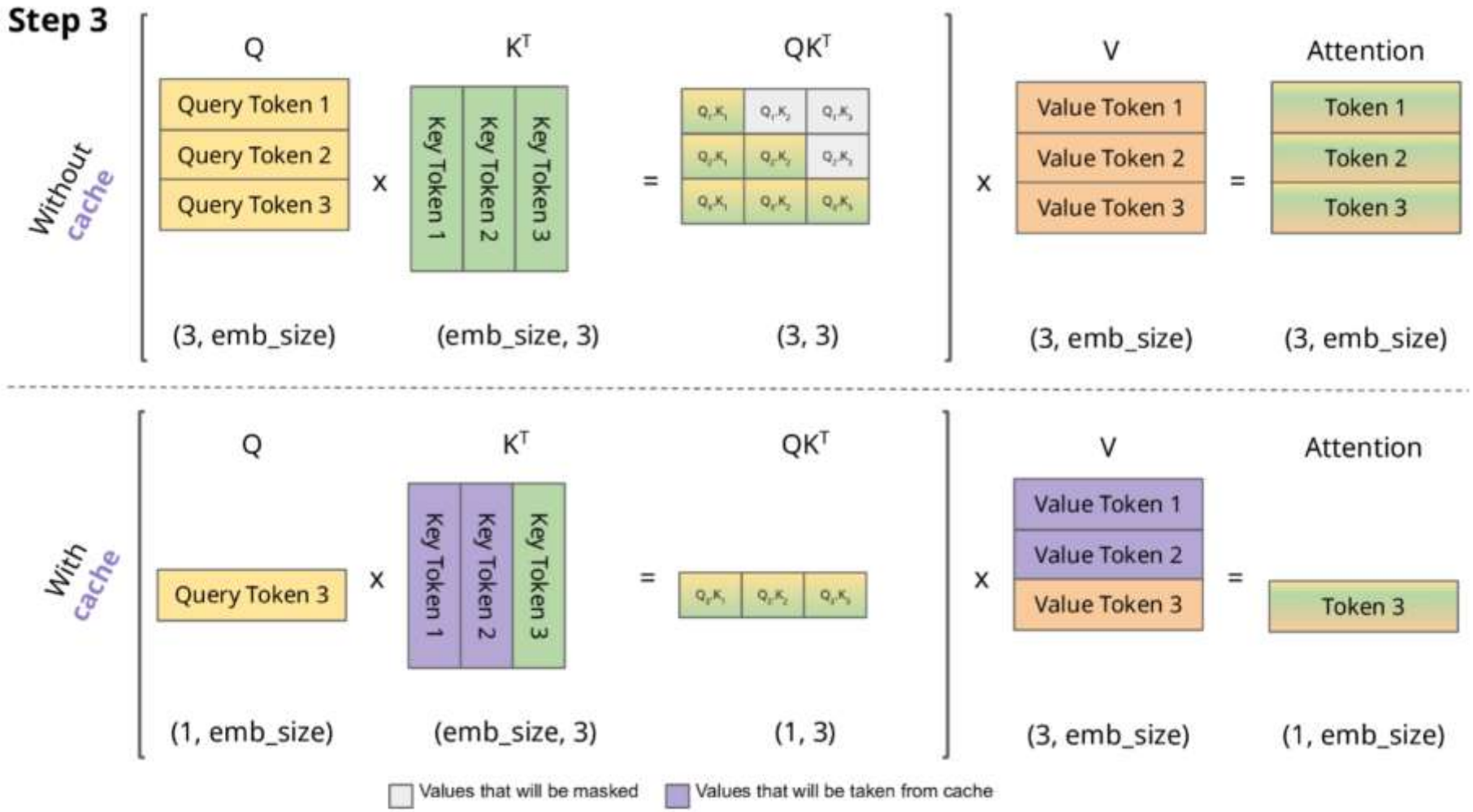


KV cache

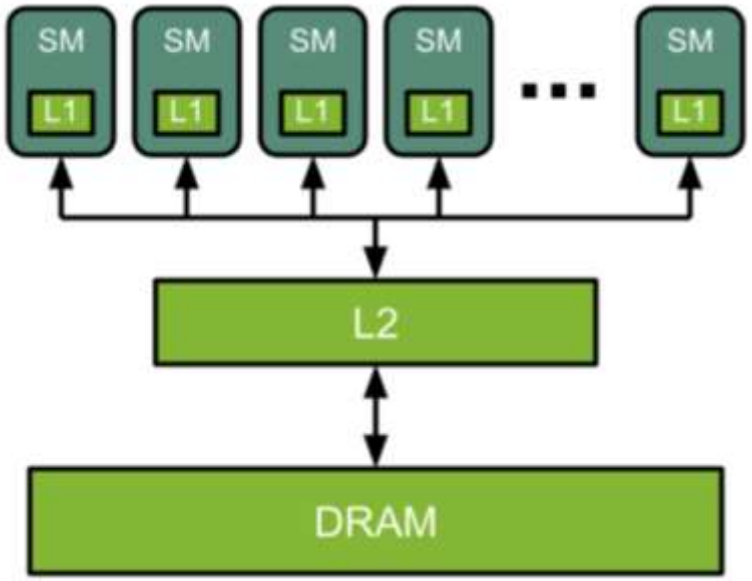
计算 Q、K、V：$Q = XW^Q, K = XW^K, V = XW^V$

计算 Attention：$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

在模型推理时，由于是自回归模式，需要一个 token 一个 token 的计算，在计算当前 token 时，前面 token 的 QKV 在之前的步骤已经算过了，不需要再次计算，只需要将其保存提供给当前步的 Attention 计算使用，这一方法称为 KV Cache。



GPU 架构



- **SM：**GPU 由多个流式多处理器（SM）组成，其中每个 SM 具有多个运算处理核心。
- **L1 Cache：**每个 SM 具有一个 L1 缓存，可以缓存 L2 缓存中经常访问的数据。
- **L2 Cache：**由所有 SM 共享，缓存全局内存中经常访问的数据以减少延迟。
- **DRAM/HBM：**高容量的 GPU 片外全局内存。

Nvidia A100：

- 108 个 SM
- fp16 张量核心：每秒 312 万亿次浮点运算（312 TFLOPS）
- DRAM 空间 80G，带宽 2.039TB/s
- L2 缓存空间 40M，带宽 4TB/s
- L1 缓存空间 192KB*108=20MB（每个 SM 192KB），带宽 19TB/s

模型推理时：

- 内存访问：从 DRAM 加载模型权重到 L2 缓存，然后传输到 SM。
- 计算：在 SM 中执行矩阵乘法，SM 请求张量核心执行计算。
- 最后将计算的结果数据写回到 DRAM 中。

从内存中读写一个数的时间，GPU 可以做近百次运算。

计算量和访存量

- **计算量：**模型进行一次完整的前向传播所发生的运算次数。浮点计算次数写作 FLOPs (Floating Point Operations)，定点计算次数常见的用 TOPS (Tera Operations Per Second)、GOPS (Giga Operations Per Second) 来表示。
- **访存量：**输入单个样本，模型完成一次前向传播过程中所发生的内存交换总量。在理想情况下（不考虑片上缓存），模型的访存量就是模型各层权重参数的内存占用（**参数访存**）与每层所输出的特征图的内存占用（**激活访存**）之和，如果用了 KV cache 方案，还需加上 **KV cache** 的大小。举例来说，权重用 fp32 存储的 7B 模型的参数访存是 7*10^9*4 Byte。

计算访存比

计算量越小，模型推理就越快吗？不是。

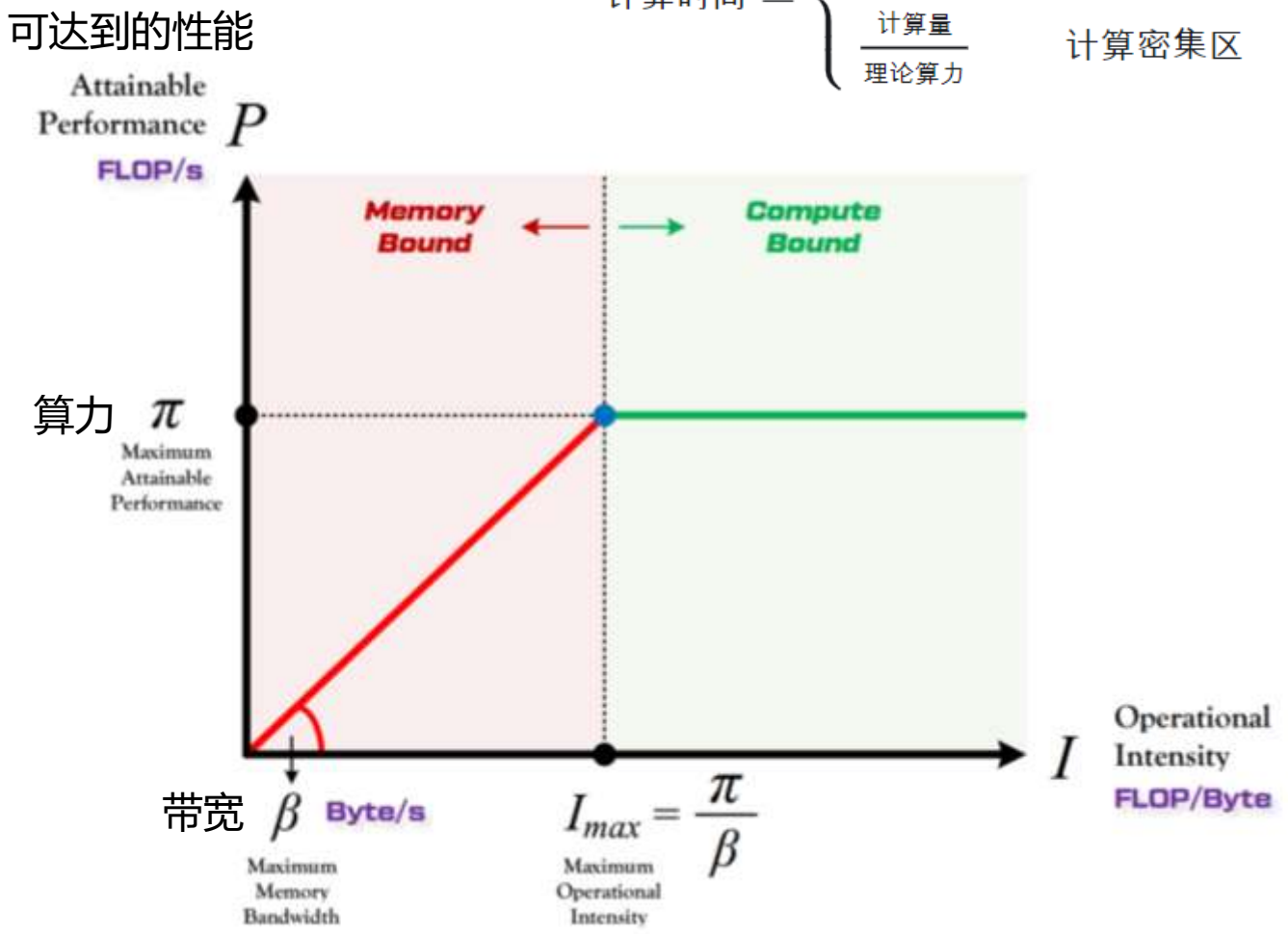
芯片的计算强度 $I_{max} = \frac{\text{算力 } \pi\ (FLOPs)}{\text{带宽 } \beta\ (Bytes)}$

模型的计算强度 $I = \frac{\text{模型计算量 } (FLOPs)}{\text{模型访存量 } (Bytes)}$

模型的理论性能 P：模型在芯片上所能达到的每秒浮点运算次数（理论值）。单位是 FLOPS。

$$P = \begin{cases} \beta \cdot I, & when\ \ I < I_{max} \quad \textbf{Memory Bound} \\ \pi, & when\ \ I \geqslant I_{max} \quad \textbf{Compute Bound} \end{cases}$$

$$\text{计算时间} = \begin{cases} \frac{\text{访存量}}{\text{带宽}} & \text{访存密集区} \\ \frac{\text{计算量}}{\text{理论算力}} & \text{计算密集区} \end{cases}$$



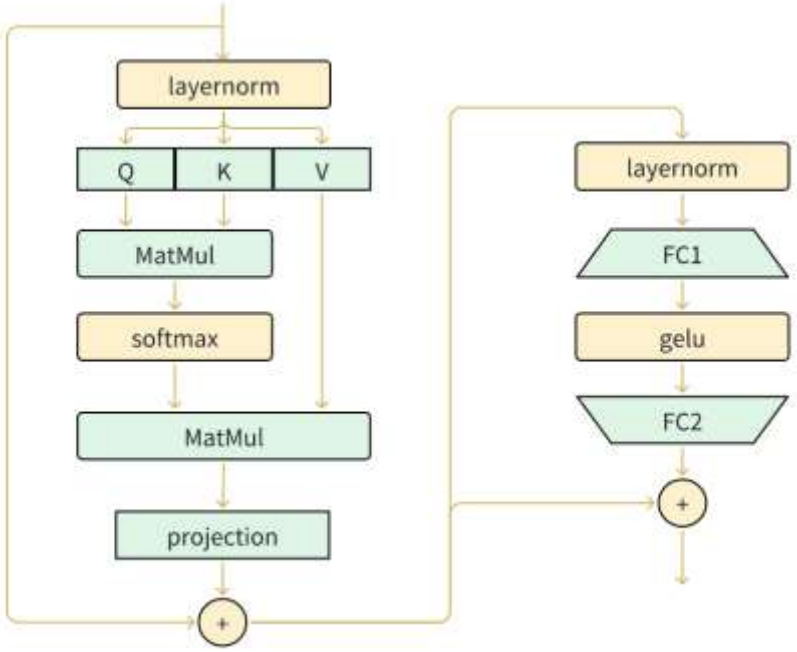
带宽瓶颈：性能卡在带宽，需要减少模型的访存量来加快推理速度。

计算瓶颈：性能卡在计算，需要减少模型的计算量来加快推理速度。

LLM 模型计算量、访存量

- 端侧 batch_size = 1，计算量大头在 QKV 和 FFN 中的 linear；访存量 90% 都是参数访存。
- 计算访存比 = 13.408 / 5.371 = 2.5

$n_{input} = 1024$	计算量	Llama2 7B 计算量
QKV	$3 * 2 * n_{heads} * (1 * d_{model} * d_{head}) = 6d_{model}^2$	100663296 = 100M
Attention	$n_{heads} * \left(2 * \left(1 * d_{head} * n_{input}\right) + 2\left(1 * n_{input} * d_{head}\right)\right) = 4n_{input}d_{model}$	16777216 = 16.8M
Projection	$2 * (1 * d_{model} * d_{model}) = 2d_{model}^2$	33554432 = 33.6M
FFN	$2 * (1 * d_{model} * (4 * d_{model}) + 1 * (4 * d_{model}) * d_{model}) = 16d_{model}^2$	268435456 = 268.4M
Total	$n_{layers} * (24d_{model}^2 + 4n_{input}d_{model})$	0,419G * 32 = 13.408G



$n_{input} = 1024$	参数访存	激活访存	KV cache 访存	总访存	7B 参数访存	7B 激活访存	7B KV cache 访存	7B 总访存
QKV	$3d_{model}^2$	$(2n_{input} + 1)d_{model}$	$2(n_{input} - 1)d_{model}$	$3d_{model}^2 + 4n_{input}d_{model} - d_{model}$	50,331,648	8,392,704	8,380,416	67,104,768
Attention	0	$3d_{model} + 2n_{\text{head}}$	0	$3d_{model} + 2n_{head}$	0	12,352	0	12,352
Projection	d_{model}^2	d_{model}	0	$d_{model}^2 + d_{model}$	16,777,216	4096	0	16,781,312
FFN	$8d_{model}^2$	$9d_{model}$	0	$8d_{model}^2 + 9d_{model}$	134,217,728	36,864	0	134,254,592
Total	$n_{layers} * 12d_{model}^2$	$n_{layers} * ((2n_{input} + 14)d_{model} + 2n_{head})$	$2n_{layers}(n_{input} - 1)d_{model}$	$n_{layers} * \left(12d_{model}^2 + 4n_{input}d_{model} + 12d_{model} + 2n_{head}\right)$	151,045,923 * 32 = 4.833G	8,446,016 * 32 = 0.27G	8,380,416 * 32 = 0.268G	5.371G

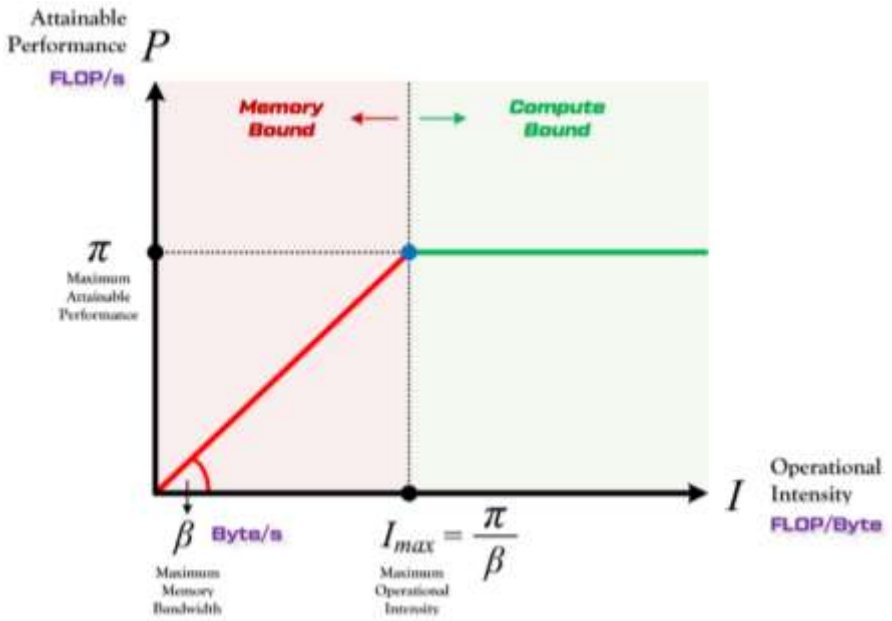
LLM 模型计算强度

- LLM 模型计算强度

解码阶段的计算密度：$\frac{n_{layers}*(24d_{model}^2+4n_{input}d_{model})}{n_{layers}*(12d_{model}^2+4n_{input}d_{model})} \approx 2$

- 3090/4090/A100 显卡的计算强度

显卡	算力 (fp16)	带宽 (GPU)	计算强度
3090	35.58 TFLOPS	936.2 GB/s	152
4090	330 TFLOPS	1 TB/s	1320
A100	312 TFLOPS	2 TB/s	624



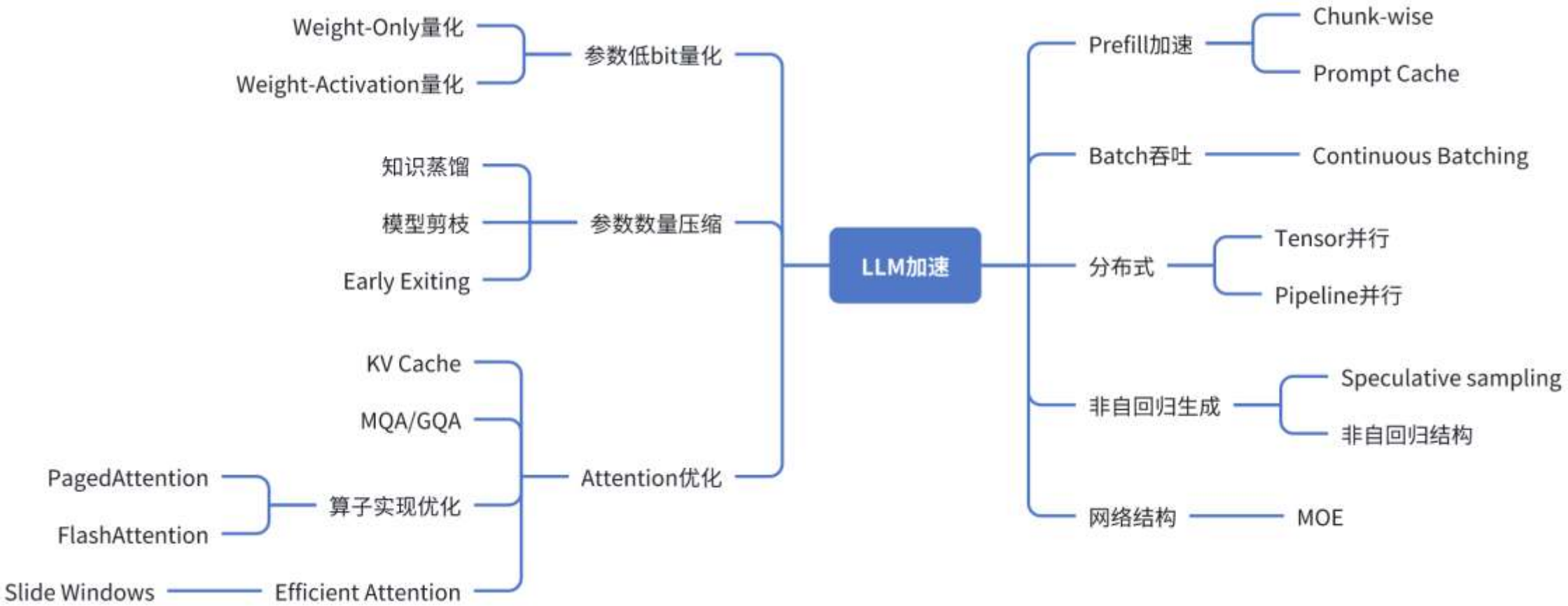
结论：LLM 模型在 decode 解码阶段都是带宽瓶颈。

推理加速的关键是：

- 降低访存量，相比于 fp16 模型和推理，理论上 int4 速度可以提升 4 倍。
- 提升计算量：增大 batch_size，但端侧场景的 batch_size 一般为 1。

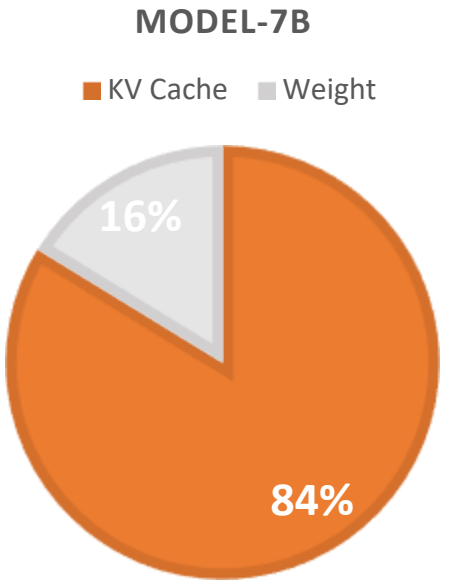
LLM 加速方法

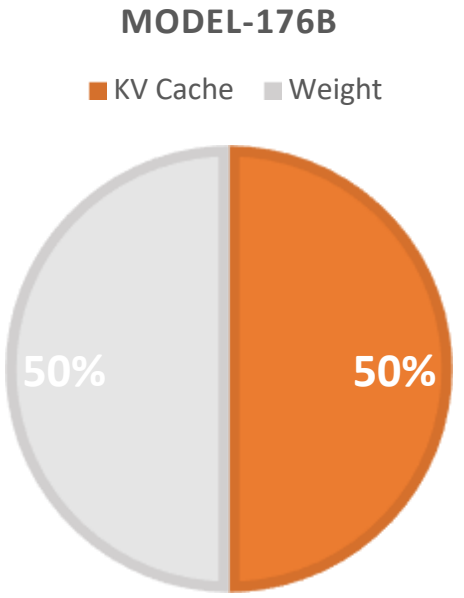
- 加速方法分类和概要



LLM 在不同场景下的优化

- 服务端
  - batch_size 越大越好，重点关注吞吐量指标。
  - 服务端小模型推理，重点是 kv cache 量化。服务端大模型推理，kv cache 和 weight 都要量化。
  - weight-only 量化在服务端较大 batch_size 时没有加速效果，该场景需要加速计算，weight-only 压缩访存是无效的。可以通过 int8 计算代替 fp16 来加速计算。
- 端侧
  - batch_size = 1，重点关注延迟指标。
  - weight 是大头，占比超过 90%，重点是 weight 量化。



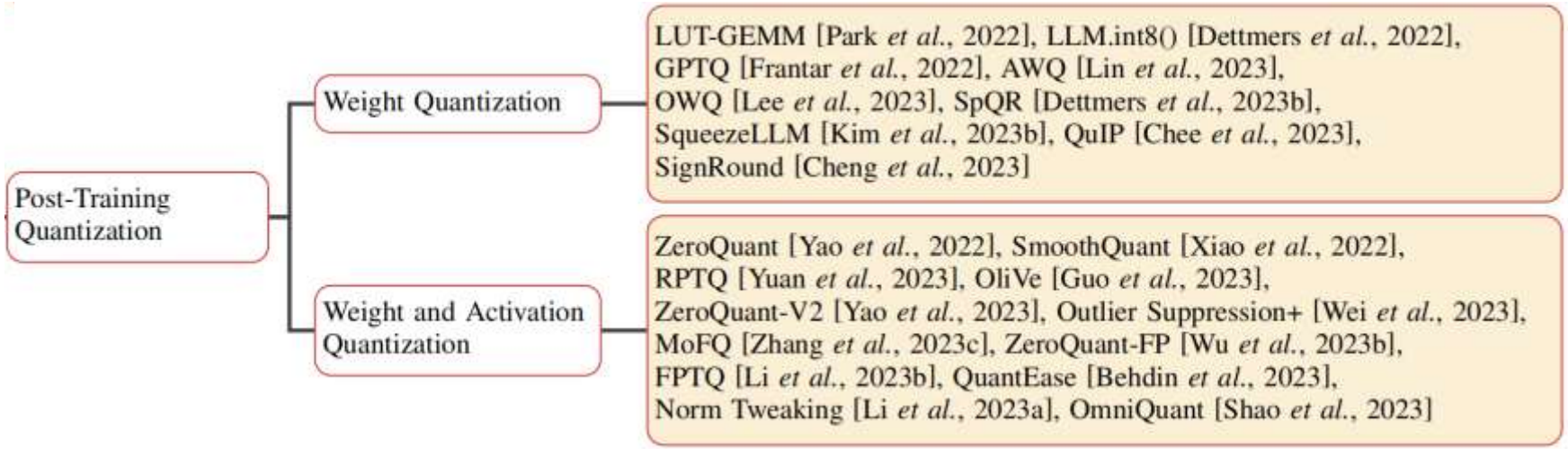


图片来源：https://pan.baidu.com/s/1hKlrHYe0BviQUopY3hUJ1g?pwd=3mv8

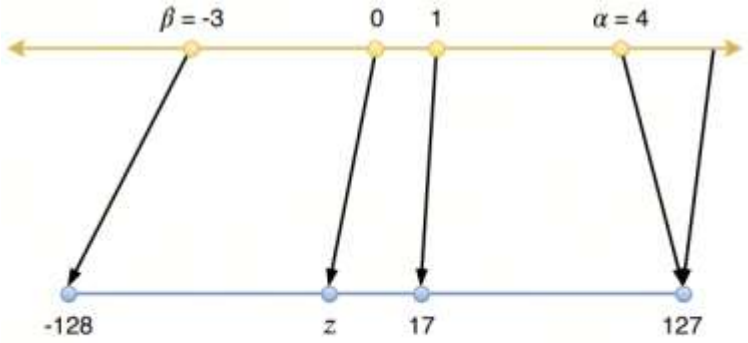
二、LLM 量化

为什么要做量化

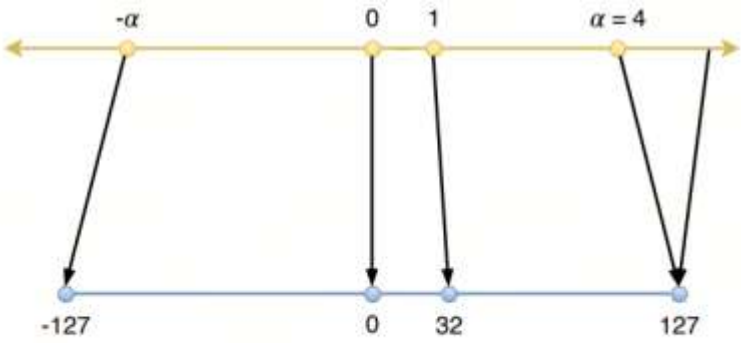
- LLM 模型主要是带宽瓶颈，需要减少访存量来加速推理。（权重量化）
  - 参数访存占用量最大，占比 90% （batch_size = 1，cache_size = 1024），最重要的是减少参数访存。需要将 fp32/fp16 的权重参数量化到更低比特位，如 int8/int4，以减少参数访存量大小。
  - 在端侧（batch_size = 1，cache_size = 1024） KV cache 访存量占比 5%，KV cache 在端侧场景下的访存优化不是重点。
- 芯片上的定点算力一般比浮点强，LLM 模型推理时考虑使用更多定点算力。（激活值量化）



量化分类



(a) Affine quantizatior 非对称量化



(b) Scale quantization 对称量化

对称量化 quantize 过程：

$$s = \frac{\alpha}{2^{b-1} - 1}$$

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

式中，s 是 scale factor，浮点数表示的范围是 [-α，α]，b 是量化位数，x 是需要量化的浮点数，x_q 是量化后的定点数。

对称量化 dequantize 过程：

$$\hat{x} = \text{dequantize}(x_q, s) = s \cdot x_q$$

以 int8 量化举例，有符号对称量化的整形范围是 [-127, 127]

-0.39	4.00	3.72	-3.00	1.56

α = 4.00

s = 4.00/127 = 0.031496

Quantize

-12	127	118	-95	50

Dequantize

-0.3780	4.00	3.7165	-2.9921	1.5748

Quantize:

-0.39 / s = -12.3825

4.00 / s = 127.0

3.72 / s = 118.11

-3.00 / s = -95.25

1.56 / s = 49.53

Dequantize:

-12 * s = -0.3780

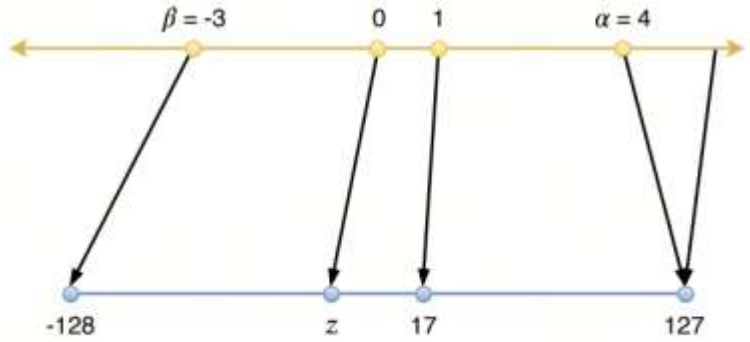
127 * s = 4.00

118 * s = 3.7165

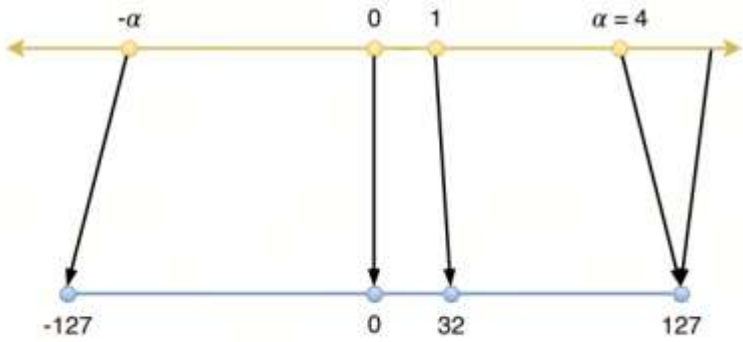
-95 * s = -2.9921

50 * s = 1.5748

量化分类



(a) Affine quantizatior 非对称量化



(b) Scale quantization 对称量化

非对称量化 quantize 过程：

$$s = \frac{\alpha - \beta}{2^b - 1}$$

$$z = -\text{round}\left(\frac{\beta}{s}\right) - 2^{b-1}$$

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

式中，s 是 scale factor，z 是 zero-point，浮点数表示的范围是 [β，α] 。

非对称量化 dequantize 过程：

$$\hat{x} = \text{dequantize}(x_q, s, z) = s \cdot (x_q - z)$$

以 int8 量化举例，有符号非对称量化的整形范围是 [-128, 127]

-0.39	4.00	3.72	-3.00	1.56

Quantize

-33	127	117	-128	38

Dequantize

-0.3843	4.0078	3.7333	-2.9922	1.5647

β = -3.00

α = 4.00

s = 7.00 / 255 = 0.027450

z = 109 – 128 = -19

Quantize:

-0.39 / s + z = -33.21

4.00 / s + z = 126.71

3.72 / s + z = 116.51

-3.00 / s + z = -128.29

1.56 / s + z = 37.83

Dequantize:

(-33 - z) * s = -0.3843

(127 - z) * s = 4.0078

(117 - z) * s = 3.7333

(-128 - z) * s = -2.9922

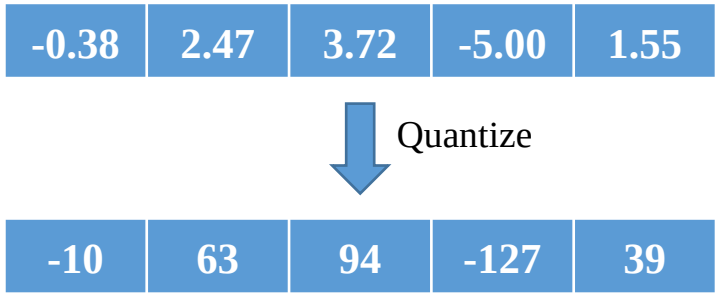
(38 - z) * s = 1.5647

矩阵乘法量化

对称量化

$\alpha_1 = 5.00$

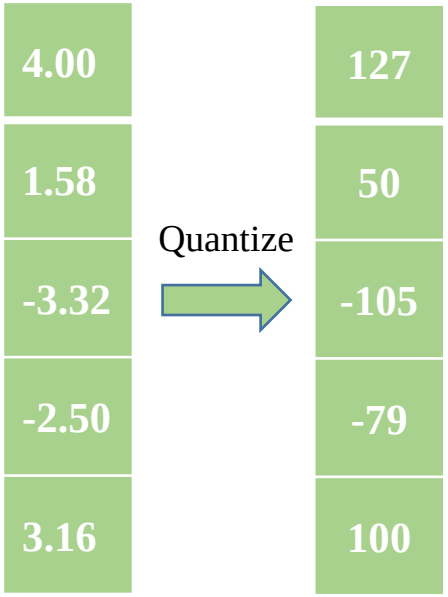
$s_1 = 5.0 / 127 = 0.039370$

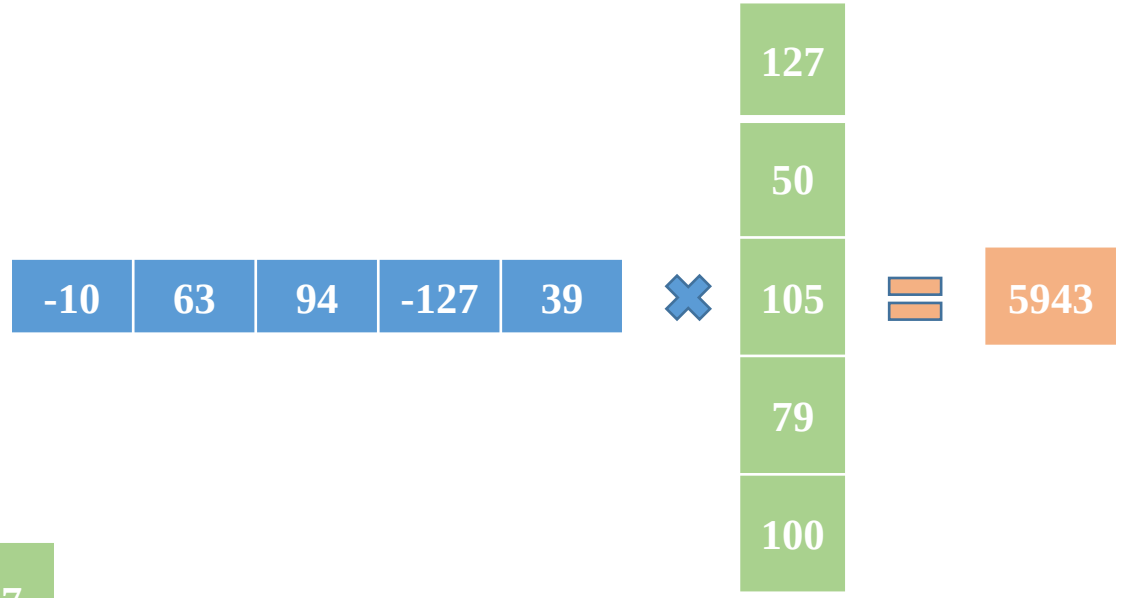


对称量化

$\alpha_2 = 4.00$

$s_2 = 4.00 / 127 = 0.031496$





最终反量化结果：

$5943 * s_1 * s_2 = 7.3693$

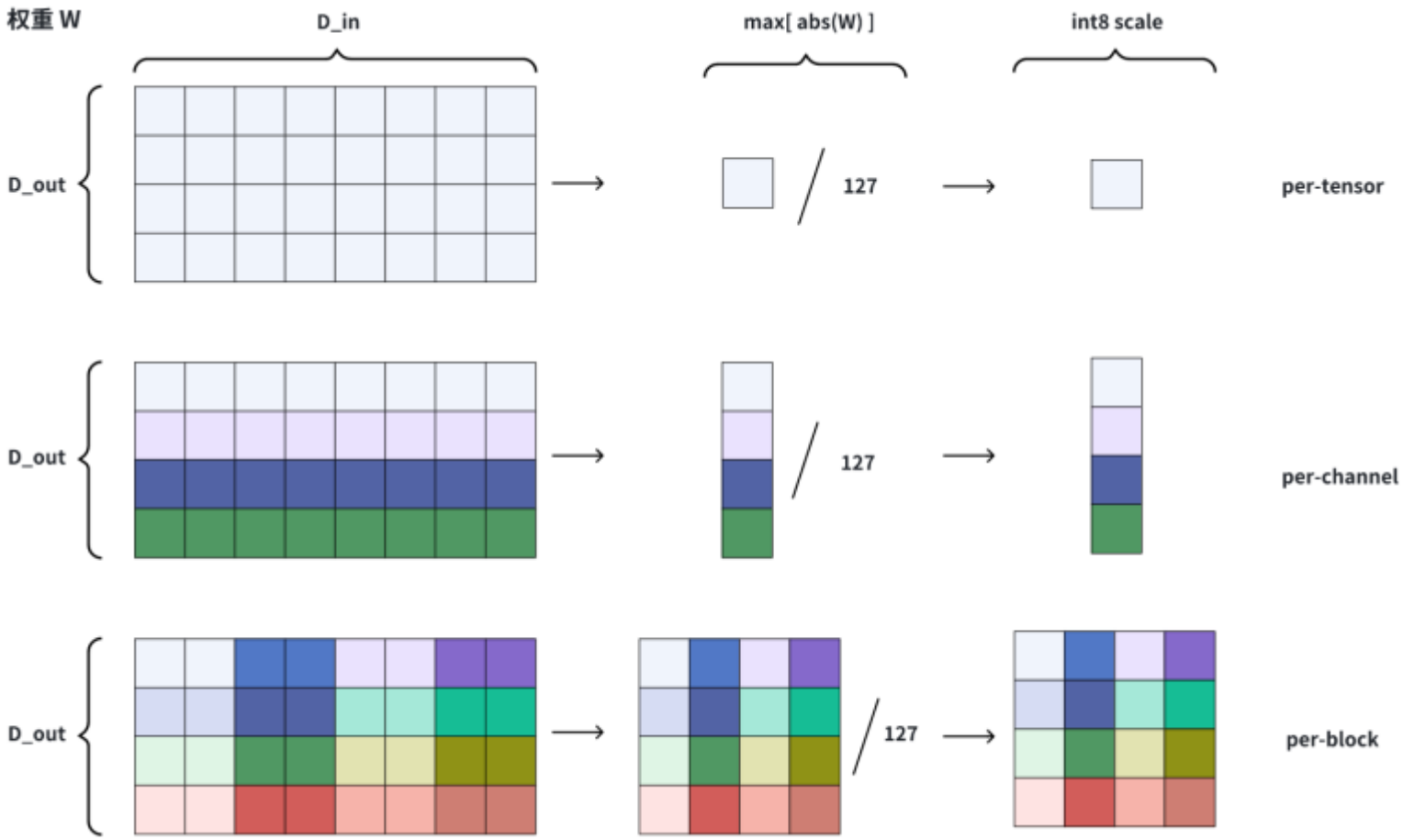
量化阶段

- PTQ (Post Training Quantization，训练后量化)：PTQ 将训练后的浮点模型直接量化为定点模型，无需对原始的浮点模型进行任何额外的训练。
- QAT (Quantization Aware Training，量化感知训练)：QAT 在模型训练时加入了伪量化节点，用于模拟模型量化时引起的误差。
  - 首先在数据集上以 FP32 精度进行模型训练，得到训练好的 baseline 模型；
  - 在 baseline 模型中插入伪量化节点，得到 QAT 模型，并且在数据集上对 QAT 模型进行 finetune；
  - 伪量化节点会模拟推理时的量化过程并且保存 finetune 过程中计算得到的量化参数；
  - finetune 完成后，使用上一步中得到的量化参数对 QAT 模型进行量化得到 INT8 模型，并部署至推理框架中进行推理。

量化粒度

对于线性层 $Y = XW^T$，输入的维度为 $[B, T, D_{in}]$，权重的维度为 $[D_{out}, D_{in}]$，输出为 $[B, T, D_{out}]$，有以下三种不同的量化粒度：

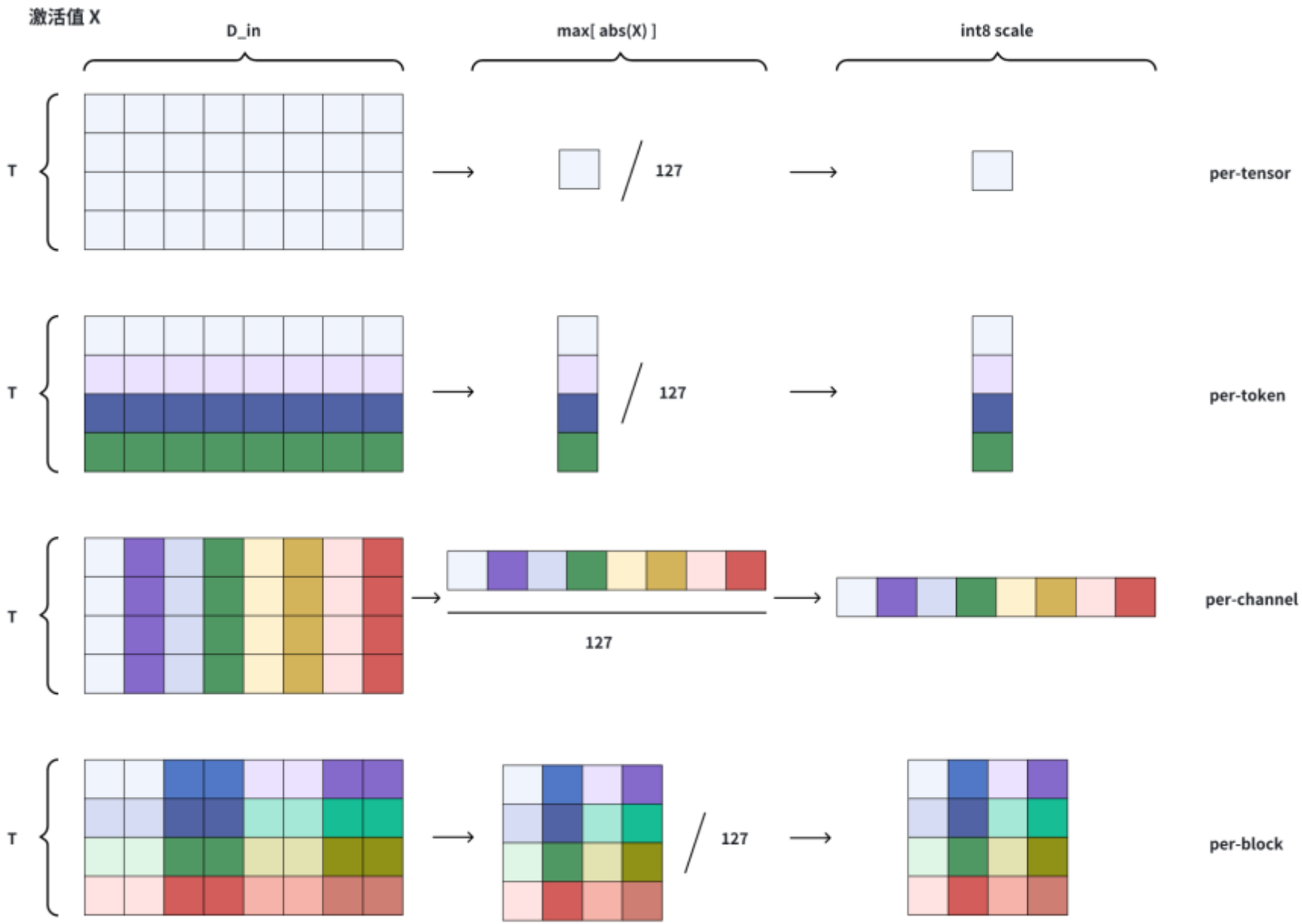
- per-tensor：整个 tensor 取绝对值的最大值来计算 scale，每个元素使用同一个 scale 量化。
- per-channel：tensor 在 D_{out} 的维度上划分为 D_{out} 个 channel，各自取绝对值的最大值来计算 scale，每个 channel 使用同一个 scale 量化。
- per-block：在 per-channel 的基础上，每个 channel 划分为 D_{in} / block_size 个 block，每个 block 使用同一个 scale 量化。



量化粒度

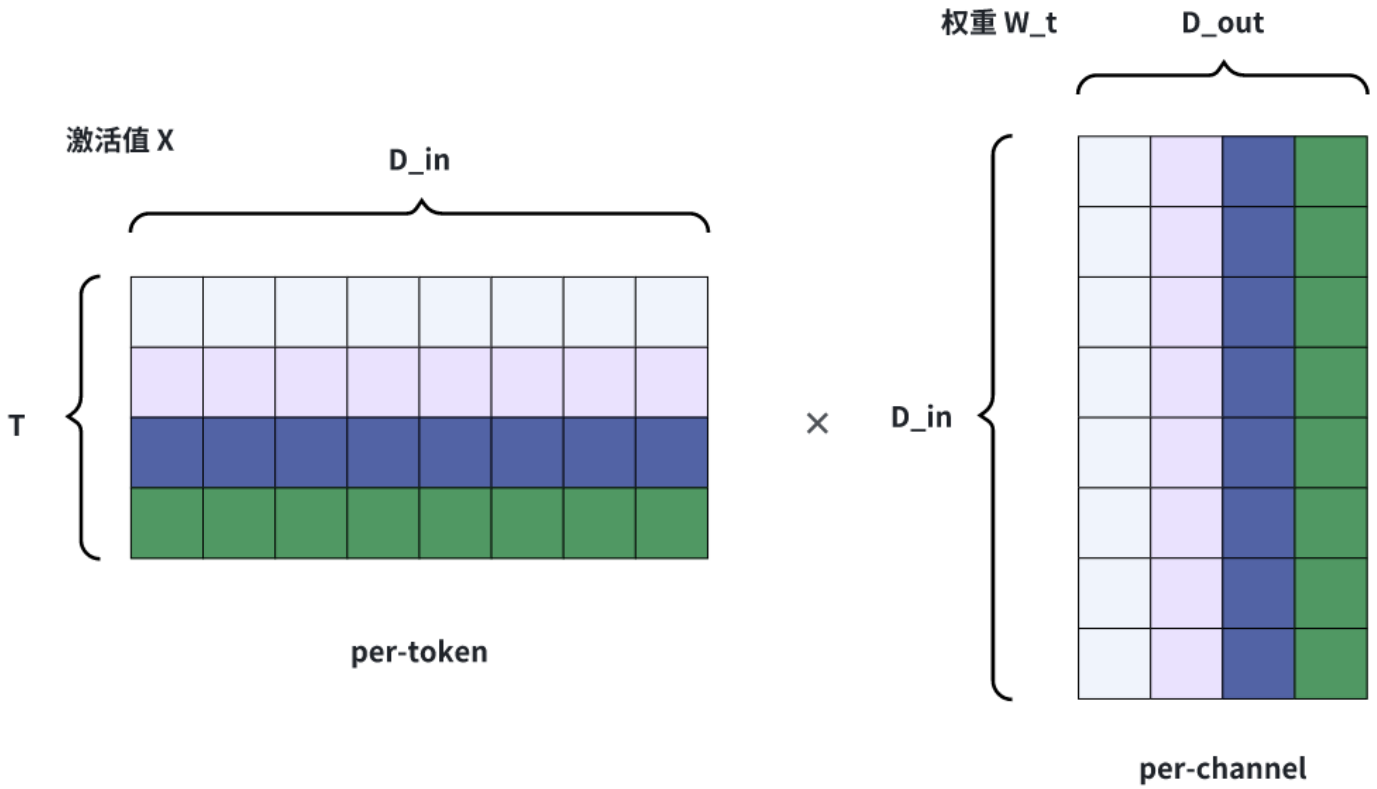
激活值的量化粒度，可以分为 per-tensor、per-token、per-channel、per-block 四种量化方式：

- per-tensor：tensor 中的每个元素使用同一个 scale 量化。
- per-token：tensor 在 T（序列长度）的维度上划分为 T 组，各自取绝对值的最大值来计算 scale，每组使用同一个 scale 量化。
- per-channel：tensor 在 D_{in} 的维度上划分为 D_{in} 个 channel，各自取绝对值的最大值来计算 scale，每个 channel 使用同一个 scale 量化。
- per-block：在 vector-wise 的基础上，每个 vector 划分为 D_{in} / block_size 个 block，每个 block 使用同一个 scale 量化。



量化粒度

per-channel 一般针对权重，per-token 一般针对激活。



量化粒度

- 激活值 A：per-tensor > per-token > per-block
- 权重 W：per-tensor > per-channel > per-block
- 常见的组合：
  - W per-tensor + A per-tensor
  - W per-tensor + A per-token
  - W per-channel + A per-tensor
  - W per-channel + A per-token
  - W per-block + A per-block
  - W per-channel + A fp16
  - W per-block + A fp16

从 input channel (D_{in}) 的维度进行量化在硬件加速中不能很好实现，因为如果从 D_{in} 矩阵乘加，需要相加的数的量化 scale 都不一样，不能直接相加。

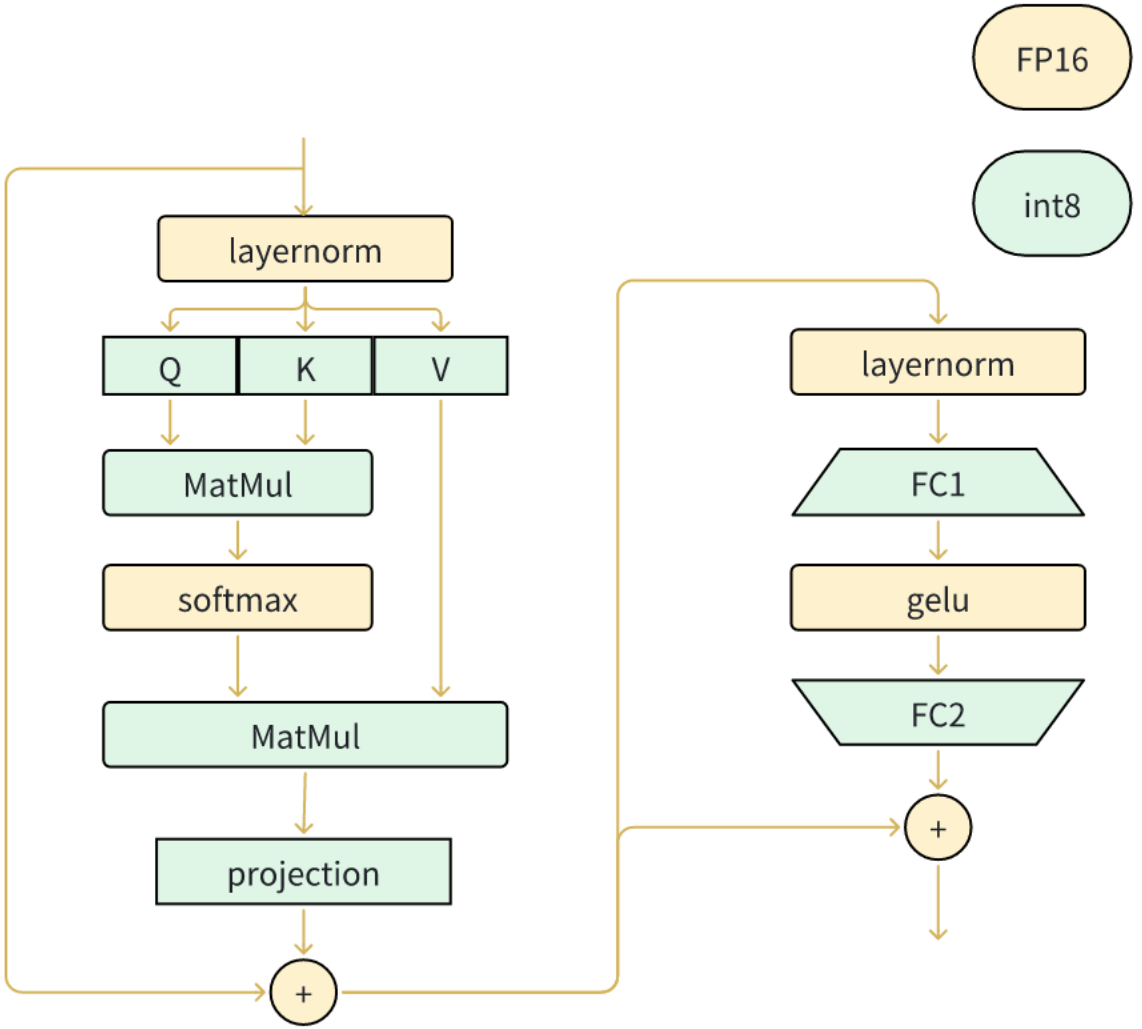
动态量化/静态量化

所有模型权重都是提前量化的，只有激活值可以提前量化，或者在推理过程中动态量化。

- 静态量化：在模型推理前提前计算好量化参数。如果运行时的数据产生的中间输出值和校准数据产生的值差别较大，则会出现很大的量化误差。
  - 使用小部分无标签校准数据对 FP32 baseline 模型进行 calibration，这一步主要是得到网络各层 weights 以及 activation 的数据分布特性（比如统计最大最小值）；
  - 根据上一步中的数据分布特性，计算出网络各层量化参数；
  - 使用上一步中的量化参数对 FP32 baseline 进行量化得到 INT8 模型，并将其部署至推理框架进行推理。
- 动态量化：在模型推理的过程中动态计算量化参数，对于不同的输入值 x，量化的参数都是推理时计算。

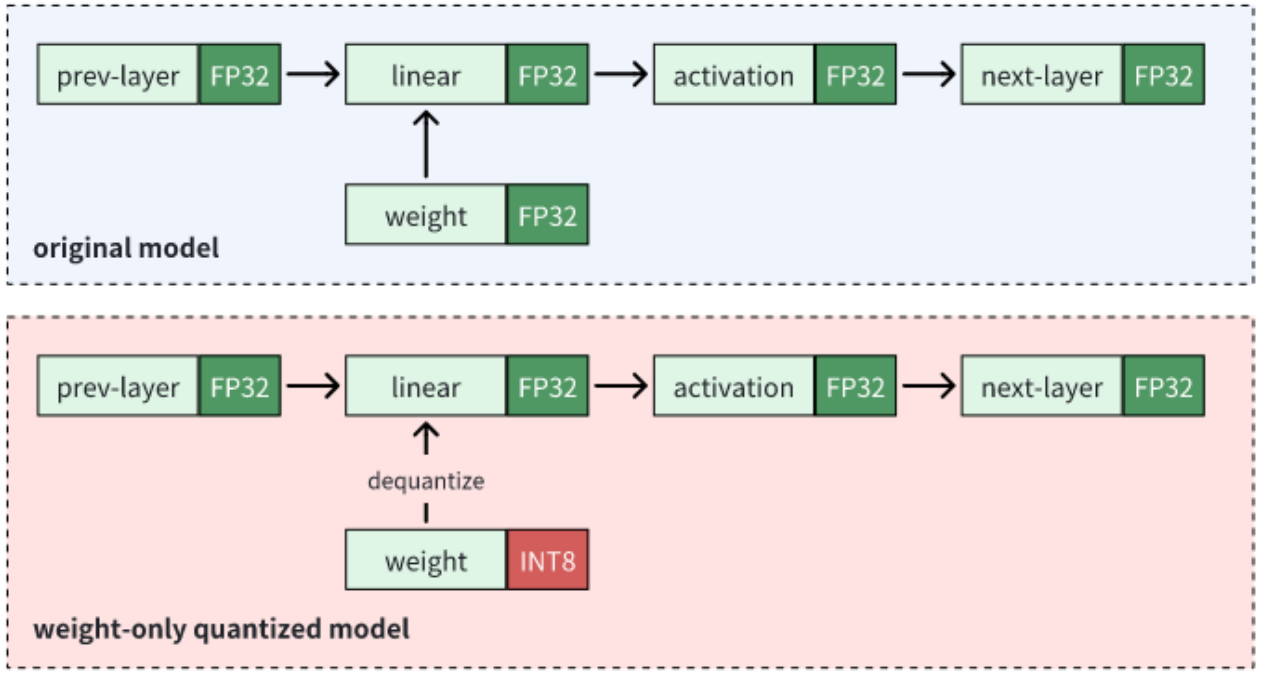
LLM 中量化的算子

- 并非所有算子都进行 int8 量化，例如，layernorm、softmax 等算子必须浮点计算，因为这些量化损失太大。
- 在实现中，一般只量化 linear 和 matmul 部分，其他部分采用浮点计算。

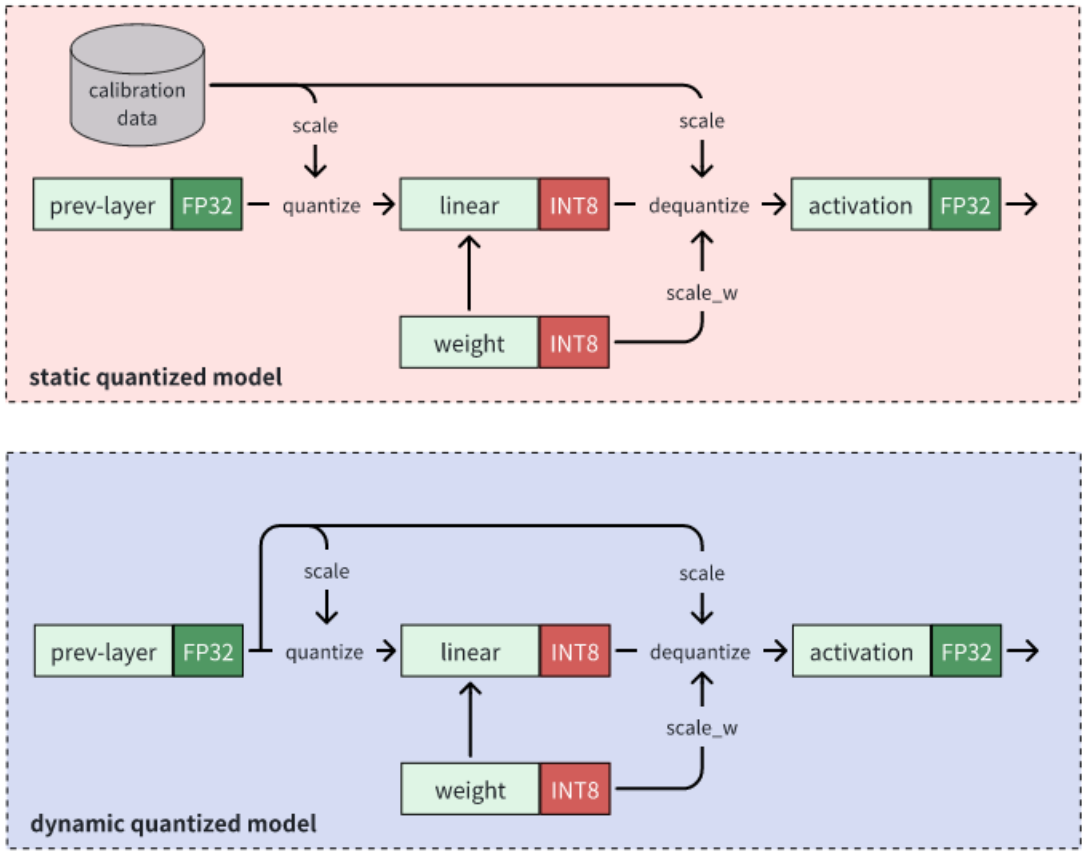


量化方案

Weight Only 量化：把权重量化为 INT8/INT4 等更低比特的类型，在模型推理时反量化为浮点数，再参与计算。适用于浮点算力较强的芯片。



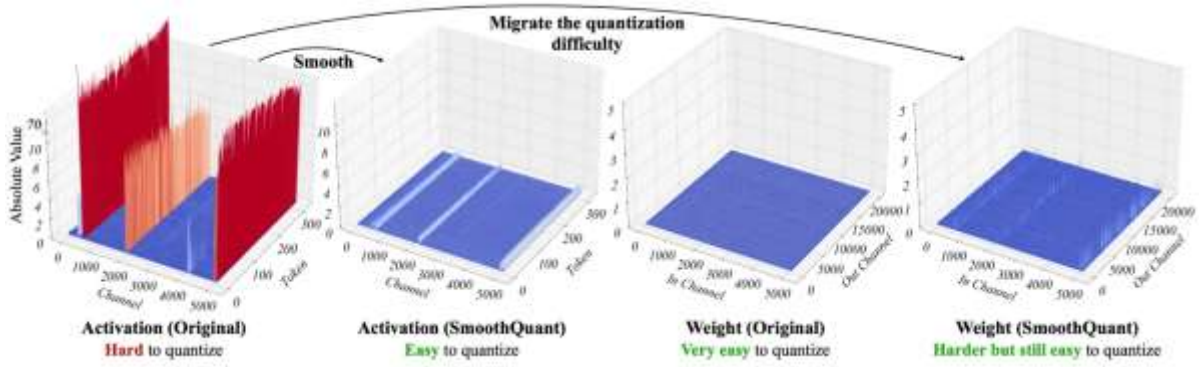
Weight + Activation 量化：提前量化权重，同时在模型推理时，将某个 op 的输入进行量化，该 op 进行定点计算。



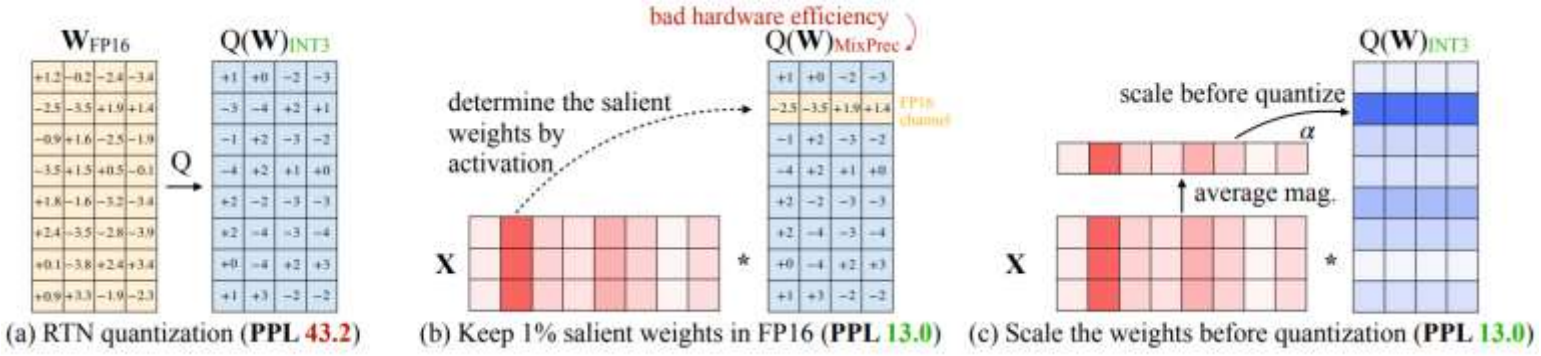
量化算法

- SmoothQuant：权重和激活值量化。由于权重很容易量化，而激活较难量化，引入平滑因子 s 来平滑激活异常值，通过数学上等效的变换将量化难度从激活转移到权重上。

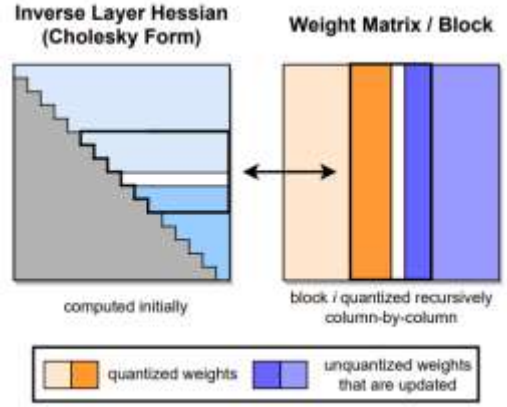
$$\mathbf{Y} = (\mathbf{X}diag(\mathbf{s})^{-1}) \cdot (diag(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$



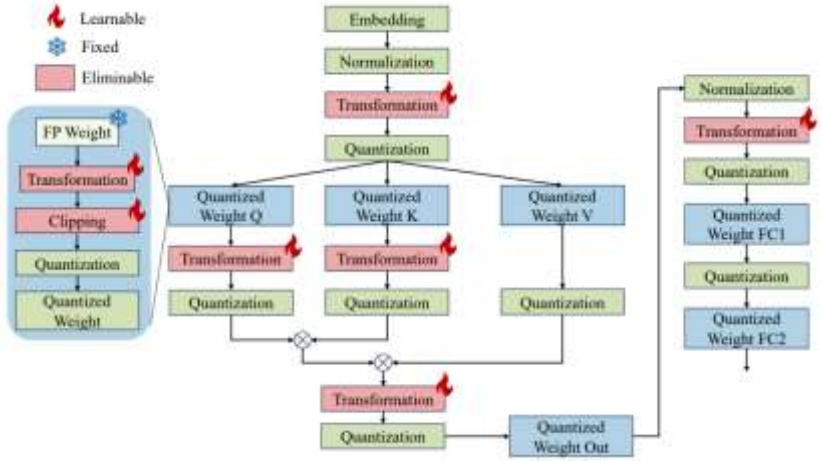
- AWQ：仅权重量化。保留 1% 的重要权重来改进 LLM 量化，通过激活感知缩放保护重要权重，将重要权重的 scale 单独再乘以一个 >1 的系数。



- GPTQ：仅权重量化。将权重分组，对每个组内的参数逐个量化后，使用校准数据算出的 Hessian 矩阵的值，更新剩余的权重（图中蓝色部分）。



- OmniQuant：可学习权重裁剪（LWC）：将 min-max 量化方法的最大值最小值进行缩放，缩放因子是可学习的，超过范围的当作离群点；可学习等效变换（LET）：类似 SmoothQuant，但平滑因子是可学习的。



LLM 量化评估指标和实验结果

- **评估指标：**
  - 使用 PPL（perplexity 困惑度）指标用于衡量量化之后的精度损失。一般 PPL 越小，量化损失越小。
- **实验结论：**
  - 量化粒度越细，精度越好。比特数越高，精度越好。
  - 权重 int4 量化，需要 per-block 级别的细粒度量化才能保证精度。权重 int8 量化，不需要 per-block 级别的量化。

量化方案	动态量化	W 量化粒度	A 量化粒度	block_size	wikitext2 ppl	量化损失
fp16	-	fp16	fp16	-	7.0768	-
W.int8 weight-only	-	per-channel	-	-	7.0848	0.11%
Wint8_Aint8	yes	per-channel	per-token	-	7.1231	0.65%
Wint8_Aint8	yes	per-channel	per-tensor	-	21.8230	208.37%
Wint4 weight-only	-	per-channel	-	-	9.7104	37.21%
Wint4 weight-only	yes	per-block	per-block	32	7.3820	4.31%
Wint4_Aint16	yes	per-channel	per-token	-	9.7149	37.28%
Wint4_Aint8	yes	per-channel	per-token	-	10.2520	44.87%
Wint4_Aint8	yes	per-block	per-block	32	7.3856	4.36%
GPTQ Wint4 weight-only	yes	per-block	-	128	8.3339	17.76%
AWQ Wint4 weight-only	yes	per-block	-	128	7.2618	2.61%

参考资料

1. Transformers KV Caching Explained： https://medium.com/@joaolages/kv-caching-explained-276520203249
2. 语言大模型推理性能工程：最佳实践：https://mp.weixin.qq.com/s/mniKrBWkDE1tWWb2wQBDCA
3. 迈向100倍加速：全栈Transformer推理优化： https://mp.weixin.qq.com/s/1QlZ_d4BrAcD9YE9BEdyYg
4. Roofline Model与深度学习模型的性能分析： https://zhuanlan.zhihu.com/p/34204282
5. 分析transformer模型的参数量、计算量、中间激活、KV cache： https://zhuanlan.zhihu.com/p/624740065
6. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation： https://arxiv.org/pdf/2004.09602.pdf
7. 大语言模型推理加速：https://www.bilibili.com/video/BV128411v7us
8. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models：https://arxiv.org/pdf/2211.10438.pdf
9. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers：https://arxiv.org/pdf/2210.17323.pdf
10. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration：https://arxiv.org/pdf/2306.00978.pdf
11. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models：https://arxiv.org/pdf/2308.13137.pdf

Q&A

- 领取课件PPT
- 参加交流群
- 获取论文

领取课件资料 截图扫码参加

《深度神经网络模型压缩》

商汤科技工程师亲授-前沿技术解析+实战教学

你将获得

- **难出其右的导师阵容**
  业界顶级模型工具链团队亲授一线实操技巧
- **即学即做，四轮项目实践**
  理论+案例分析+Project实践+讲评反馈
- **真正做到知识下限低，但上限高**
  核心知识深入浅出，一言不合就上代码
  工业界压缩和部署项目经验，直接分享
- **全程答疑+1v1代码批改，你只需负责学！**

担心课程质量低？买贵了？

开班7天内可无理由退款

课程价格699，现在前150名可享
立减50元超前优惠，仅售 ¥649元！

扫码咨询
了解课程详情