

LLMC: 大语言模型压缩工具的开发实践



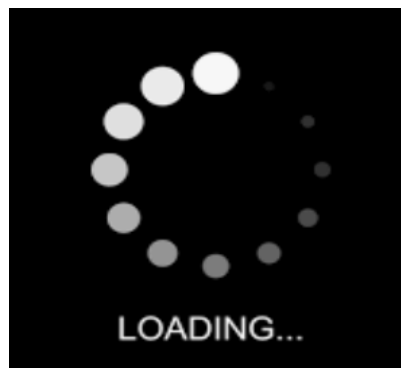
<https://github.com/ModelTC/llmc>

主讲人：谷石桥 雍洋

日期：2024年12月16日

目录

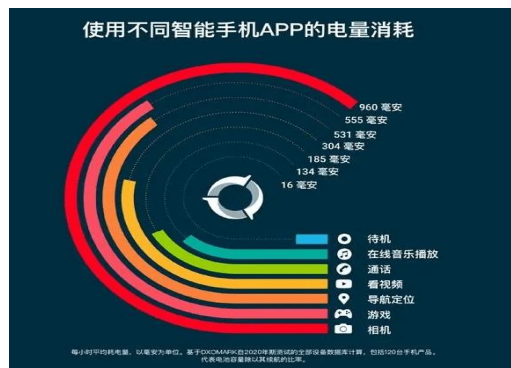
- 大模型压缩背景与挑战
- LLMC：大模型压缩工具
- LLMC的使用方法与操作指南
- LLMC的可扩展性与应用前景



高延迟引发的危险结果和糟糕用户体验

```
RuntimeError: CUDA out of memory. Tried to allocate 14.56 GiB (GPU 0: 16.00 GiB total capacity; 14.56 GiB already allocated; 0.44 GiB free; 0.00 GiB reserved for system use). If reserved memory is greater than allocated, add --max-memory-per-gpu=16000 to the command line. See documentation for MallocOptions and cudaMallocOptions for more information.
```

过高的峰值内存/显存对硬件需求也更高



高功耗带来的移动设备的电池负载

model-00003-of-00082.safetensors	1.74 GB	LFS	⬇
model-00004-of-00082.safetensors	1.74 GB	LFS	⬇
model-00005-of-00082.safetensors	1.74 GB	LFS	⬇
model-00006-of-00082.safetensors	1.74 GB	LFS	⬇
model-00007-of-00082.safetensors	1.74 GB	LFS	⬇

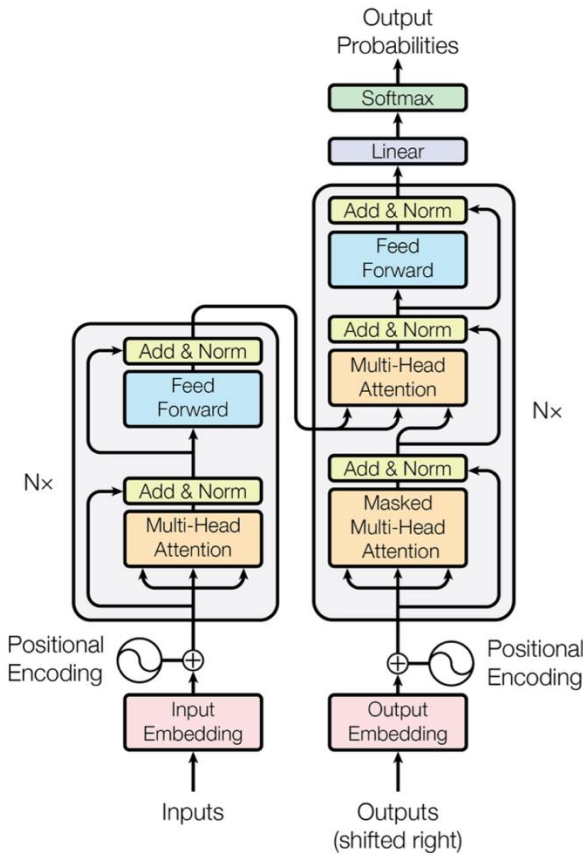
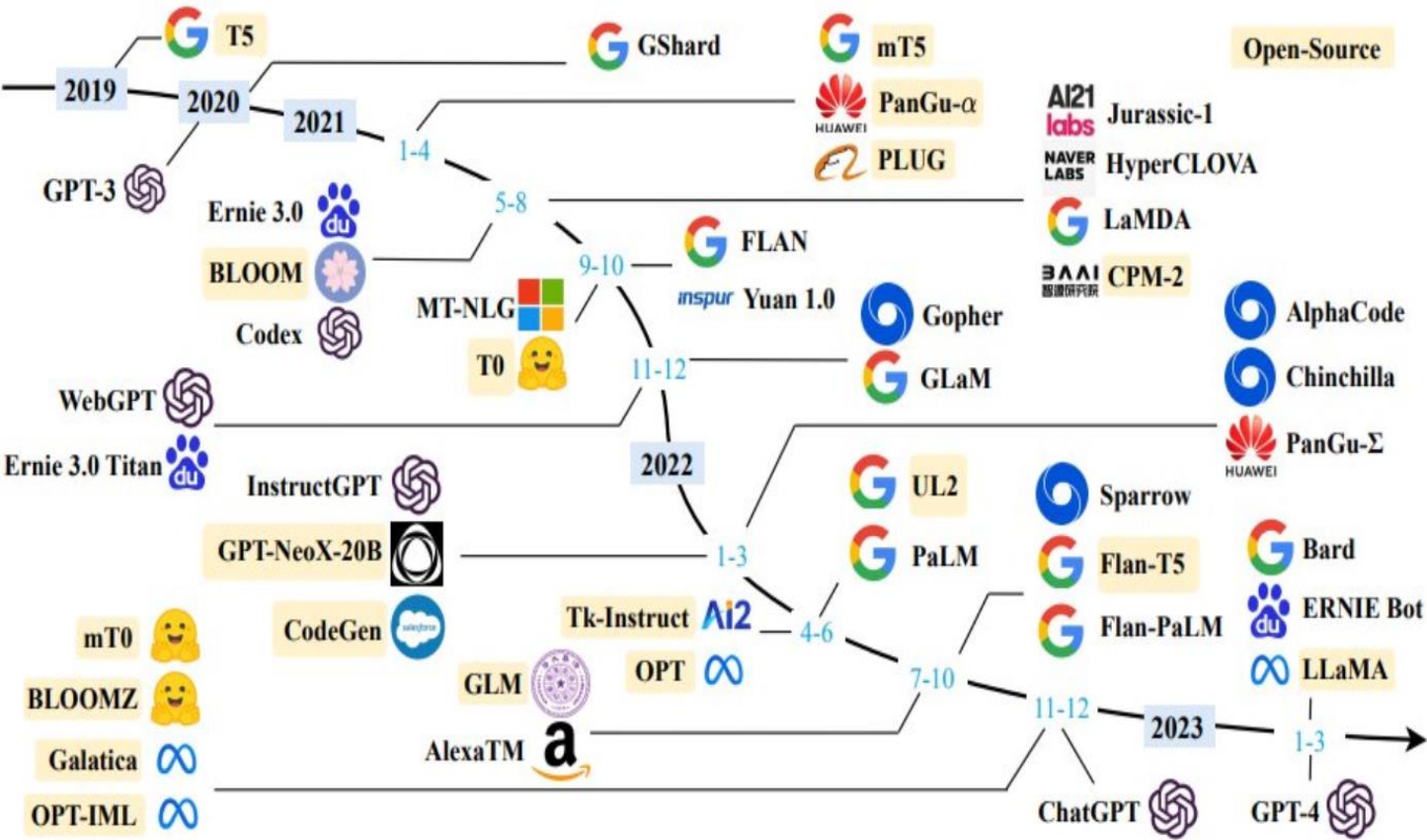
模型越大存储和传输的压力越高



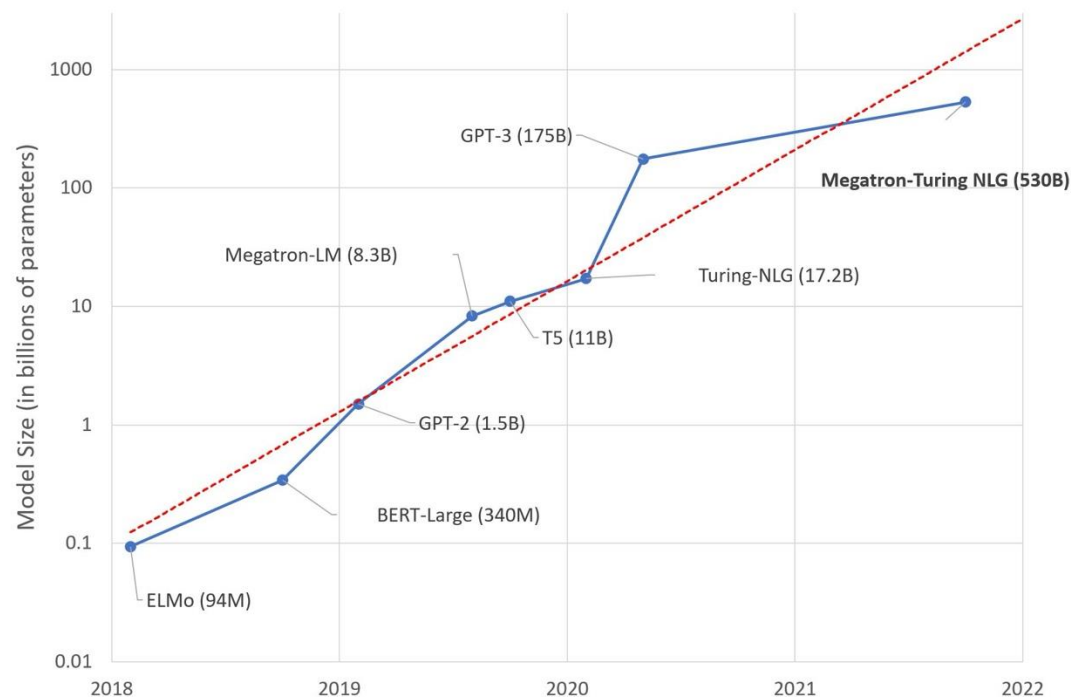
模型压缩的意义

- 提高**速度**，降低时延
- 降低**功耗**，增加续航
- 降低峰值显存，减少部署**成本**
- 压缩**体积**，优化存储和传输效率

常见的大语言模型



大语言模型参数量



部署一个175B GPT-3至少需要:

➤ 80GB NVIDIA A100 GPUs

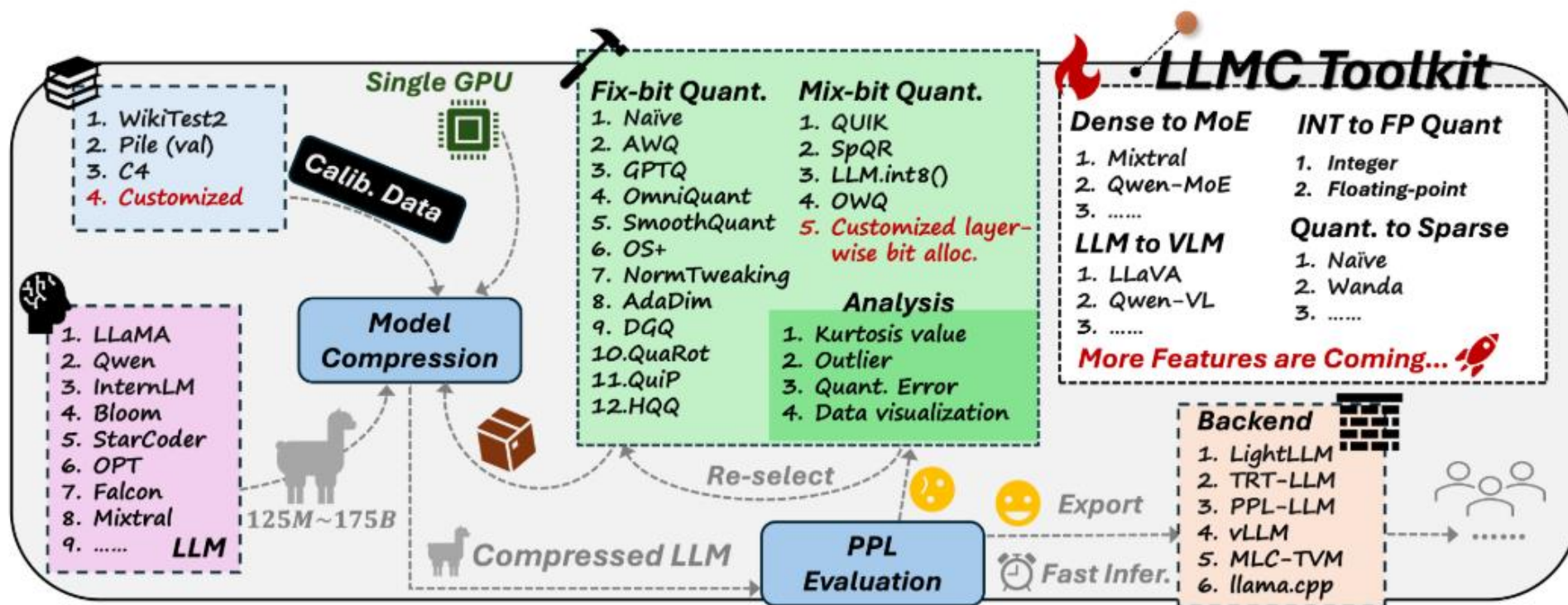


➤ 40GB NVIDIA A40 GPUs



模型体积越来越大，硬件资源有限，需要模型压缩加速推理，但是同时面临精度风险

LLMC: Towards Accurate and Efficient LLM Compression



代码链接:

<https://github.com/ModelTC/llmc>

文档链接:

<https://llmc-zhcn.readthedocs.io/en/latest/>

License Apache 2.0 LLMC 2405.06001 Star 348 visitors 30,488 chat 6 online QQ docs English 文档 中文

- 支持十八种算法，五个推理后端，数个模型结构，多种评测方式的**大模型压缩(量化+稀疏)工具包**
- **主要介绍模型量化**

LLMC框架

模型支持

基模型/对话模型/代码模型(Base/Chat/Code)
Llama, Qwen, Intern等

混合专家模型(MOE)
Deepseek, Mixtral, Qwen2moe等

多模态模型(VLM)
Llava, Qwen2VL, Llama3.2等

其他模型
Smollm, Phi, Minicpm等

算法领先

权重量化, 激活量化
AWQ, GPTQ, Quarot, Omniquant等

KVcache量化
Naive, KIVI等

混合精度量化
OWQ, QUIK, llm.int8(), Spqr等

结构化/非结构化稀疏
Sparse-GPT, Short-GPT等

核心功能

模型量化: 减少推理时延

体积压缩: 减少存储占用

精度调优: 挽救压缩模型精度

多后端导出

LIGHT LLM

TensorRT-LLM

SGL

*v*LLM

MACHINE LEARNING
COMPILATION

mlc-llm

量化评估

Perplexity



OpenCompass



lm-evaluation-harness

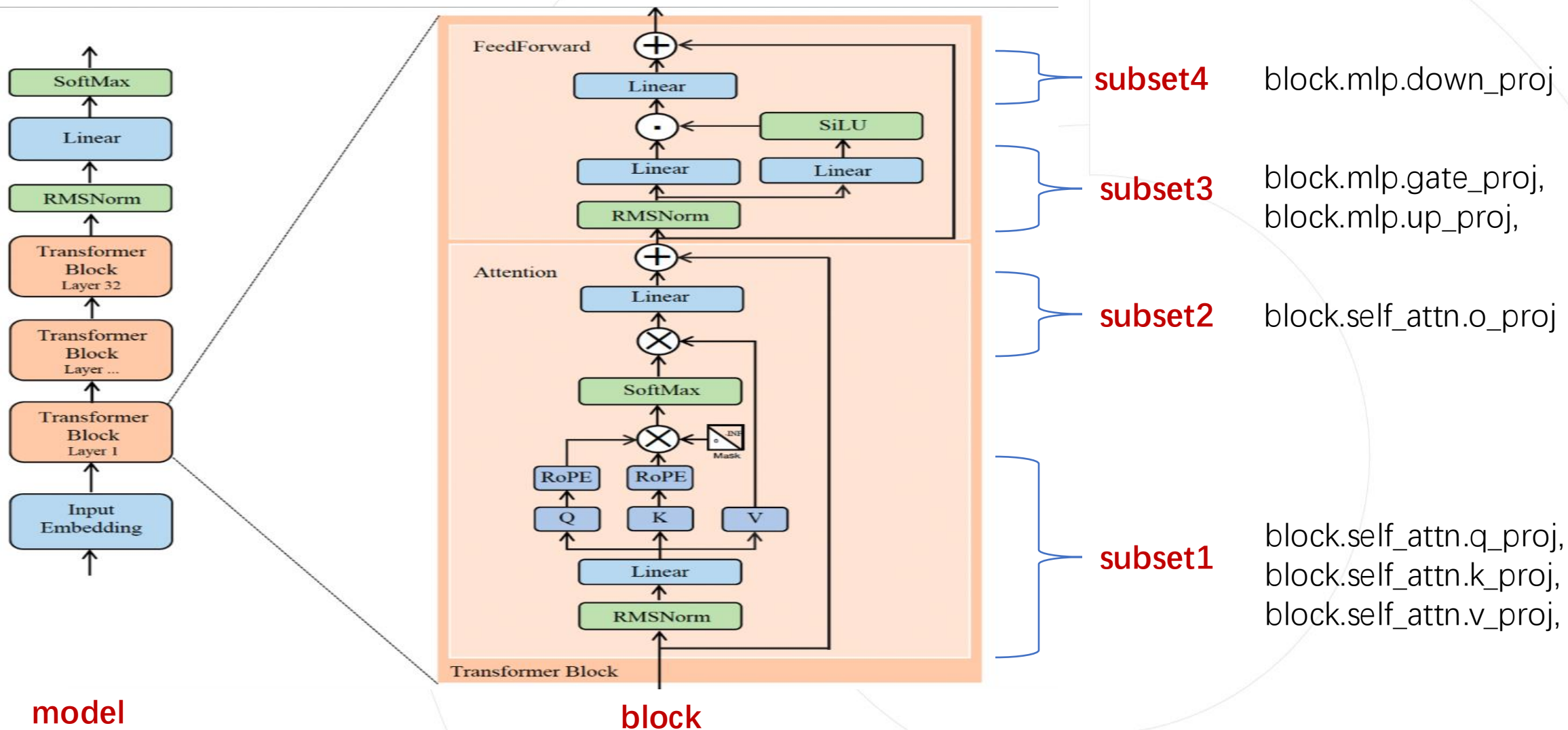
LMMs-Eval

与业界其他框架对比

Toolkit	Algorithms	Model family	Evaluation	Backends	Institution
LMQuant	AWQ, Atom, GPTQ, QoQ, QuaRot, SmoothQuant	Mixture-of-Expert, (e.g. Mixtral), Transformer-like (e.g. Llama)	Perplexity, Throughput	QServe	MIT EECS
LLMC	AWQ, AdaDim, DGQ, GPTQ, FP8, HQQ, LLM.int8(), NormTweaking, OS+, OWQ, OmniQuant, QUIK, SmoothQuant, SpQR, Quarot, Combinations	Mixture-of-Expert, (e.g. Mixtral), Transformer-like (e.g. Llama), Multi-modal (e.g. LLaVA)	Perplexity, OpenCompass ³⁶ , lm-evaluation-harness ³⁷	TensorRT-LLM, MLC-TVM, vLLM, LightLLM, Sglang, Lmdeploy, Transformers	Beihang & SenseTime
MI-optimize	AWQ, FP8, GPTQ, QuIP, RTN, SmoothQuant, SpQR, ZeroQuant, Combinations	Llama, Chatglm, Baichuan	BOSS (Robust), Perplexity, lm-evaluation-harness ³⁷	Transformers	TsingmaoAI
QLLM-Eval	AWQ, SmoothQuant	Mixture-of-Expert, (e.g. Mixtral), Transformer-like (e.g. Llama), Multi-modal (e.g. LLaVA), Long-Context (e.g. Longchat), Others (e.g. Mamba)	OpenCompass ³⁶ , lm-evaluation-harness ³⁷ , LongEval ³⁸ , Lost-in-the-middle ³⁹ , MT-Bench ⁴⁰	Transformers	Tsinghua University
LLM Compressor	GPTQ, SmoothQuant, FP8	Mixture-of-Expert, (e.g. Mixtral) Transformer-like (e.g. Llama)	-	vLLM	Neural Magic

- 支持**更多**算法
- 支持**更广泛**的模型结构
- 支持**更多**量化评估方式
- 支持**更多**推理后端

模型结构定义: **model** 包括多个 **block**, 每个 **block** 包括多个 **subset**, 每个 **subset** 包括多个 **nn.Linear**



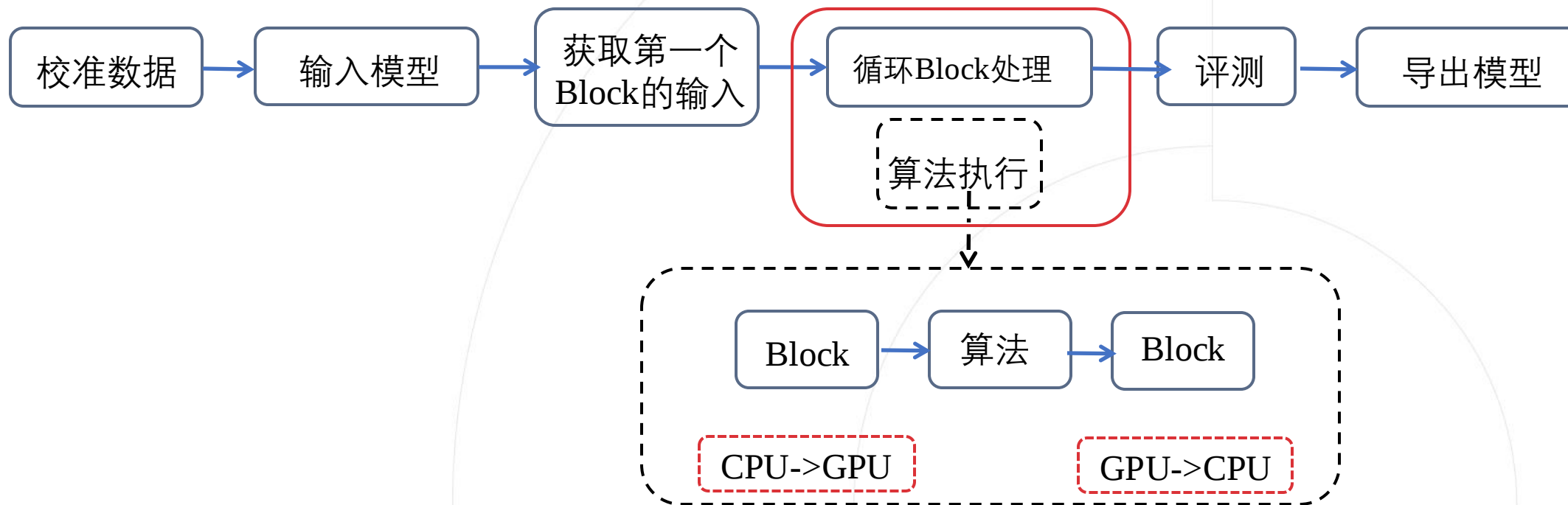
model包括多个block，每个block包括多个subset，每个subset包括多个nn.Linear

llmc / llmc / models / llama.py

```
class Llama(BaseModel):
    def get_layernorms_in_block(self, block):
    def get_subsets_in_block(self, block):
    return [
        {
            'layers': {
                'self_attn.q_proj': block.self_attn.q_proj,
                'self_attn.k_proj': block.self_attn.k_proj,
                'self_attn.v_proj': block.self_attn.v_proj,
            },
            'prev_op': [block.input_layernorm],
            'input': ['self_attn.q_proj'],
            'inspect': block.self_attn,
            'has_kwargs': True,
        },
        {
            'layers': {'self_attn.o_proj': block.self_attn.o_proj},
            'prev_op': [block.self_attn.v_proj],
            'input': ['self_attn.o_proj'],
            'inspect': block.self_attn.o_proj,
            'has_kwargs': False,
        },
        {
            'layers': {
                'mlp.gate_proj': block.mlp.gate_proj,
                'mlp.up_proj': block.mlp.up_proj,
            },
            'prev_op': [block.post_attention_layernorm],
            'input': ['mlp.gate_proj'],
            'inspect': block.mlp,
            'has_kwargs': False,
            'is_mlp': True,
        },
        {
            'layers': {'mlp.down_proj': block.mlp.down_proj},
            'prev_op': [block.mlp.up_proj],
            'input': ['mlp.down_proj'],
            'inspect': block.mlp.down_proj,
            'has_kwargs': False,
            'is_mlp': True,
        },
    ]
]
```

- 无需重写模型定义，方便开发
- 方便复用，例如大多开源模型结构都和Llama一样
- 方便扩展新模型，只需要新增subset定义即可
- 模型结构定义可参考: llmc/llmc/models/

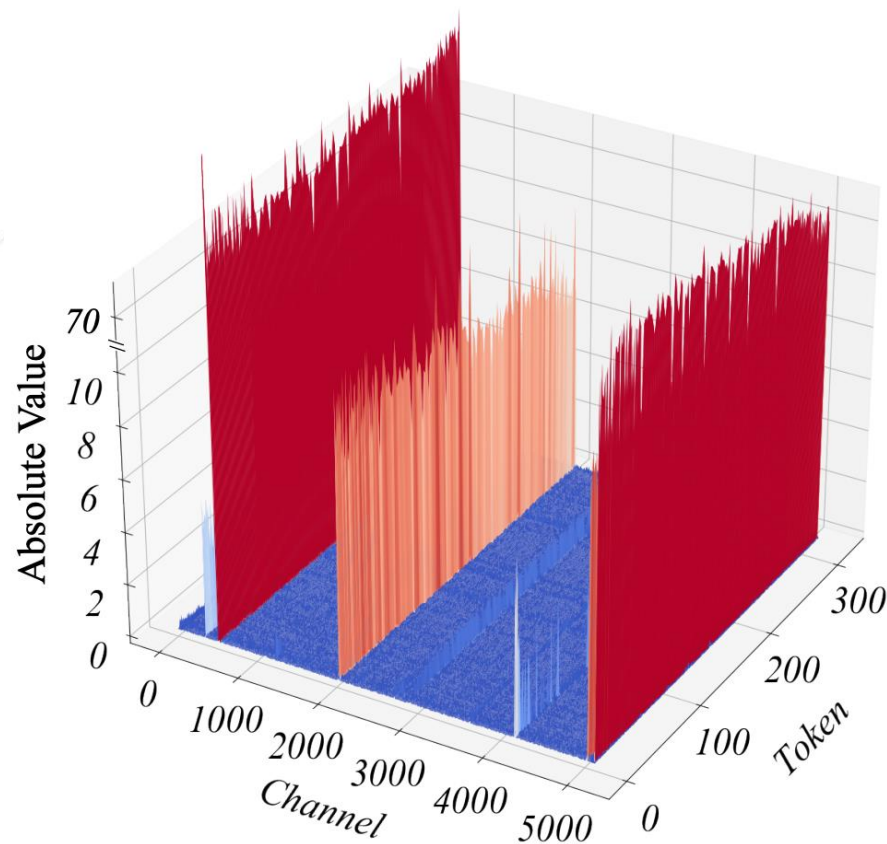
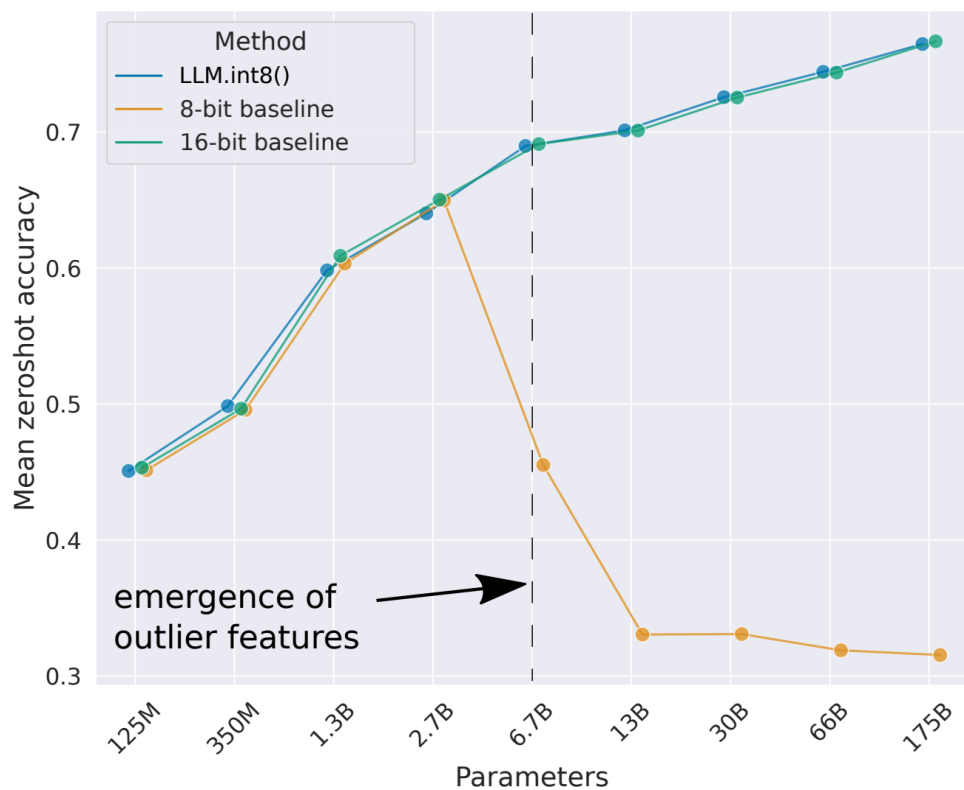
model包括多个block，每个block包括多个subset，每个subset包括多个nn.Linear



- 每个算法都会逐block执行，在block内部是逐subset执行
- GPU上每个时刻只会存在某一个block，方便运行大的模型
- 例如单个80G的GPU即可运行Llama3.1-405B，Deepseekv2-236B

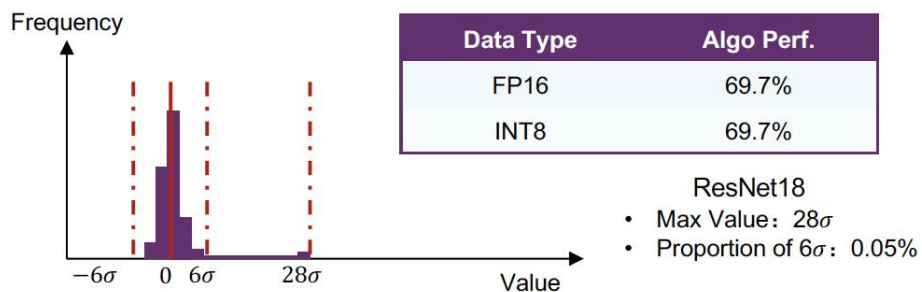
#Bit	Method	LLaMA-3.1-8B	LLaMA-3.1-70B	LLaMA-3.1-405B
w8a8	BF16	6.24	2.81	1.44
	Naive	6.36	26.34	19.12
	SmoothQuant	6.35	3.08	1.50
	OS+	6.37	183.32	1.49
	OmniQuant	6.37	-	-
w4a16	QuaRot	6.26	2.86	1.46
	Naive	6.83	3.51	1.62
	AWQ	6.64	3.27	1.60
	GPTQ	6.63	3.41	OOM
	HQQ	6.87	5.29	1.59

不同于CNN模型或者小型Transformer模型，LLM的矩阵乘法产生的激活张量通常有较多的离群值(outliers)

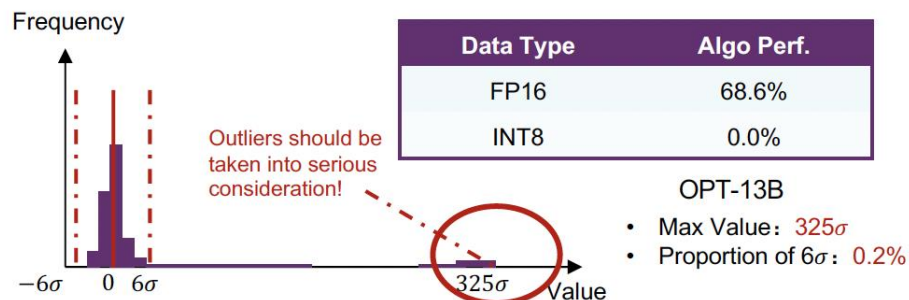


如何取舍outliers通常是量化工作中的一大难点

- 若过分考虑，则会因量化范围过大而降低量化的表达范围；
- 若过分截断，通常会因这些绝对值较大的值，在模型推理中对结果有较大影响，而导致模型效果变差，而这个现象在LLM的量化则尤为明显



→ ResNet-18, int8无损



→ OPT-13b, int8精度崩掉

- 针对outlier问题，业界提出了一系列的量化算法

精度对齐: 工具所支持的各种算法, 精度基本对齐了官方实现

Method	w4g128	w3g128	w2g64
GPTQ	5.62	6.32	14.97
GPTQ-LLMC	5.62	6.32	14.97
AWQ	5.60	6.24	2.16e5
AWQ-LLMC	5.60	6.24	2.16e5
OmniQuant	5.59	6.09	9.53
OmniQuant-LLMC	5.59	6.09	9.53

Method	LLaMA-2-7b	LLaMA-2-70b	LLaMA-3-8b	LLaMA-3-70b
Wanda	6.91	4.22	9.56	OOM
Wanda-LLMC	6.91	4.19	9.58	5.75

Method	w8a8	w6a6	w4a4
OmniQuant	5.49	5.70	12.21
OmniQuant-LLMC	5.49	5.70	12.23
Quarot w/ GPTQ.	5.48	5.50	6.22
Quarot-LLMC w/ GPTQ.	5.48	5.50	6.24

Method	w8a8
SmoothQuant	5.589
SmoothQuant-LLMC	5.589
OS+	5.511
OS+-LLMC	5.517

AWQ

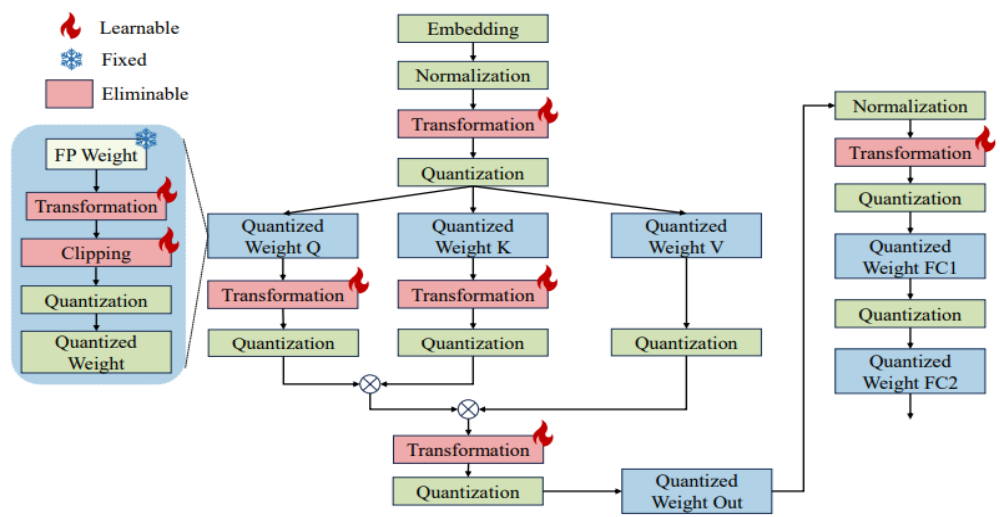
- **思想**: Weight的量化过程考虑Activation的值得影响。AWQ认为Activtion的值较大的通道对应的weight相对重要，反之则相对不重要，进而通过乘以一个缩放系数 Δ 去体现其重要性。weight量化时，对weight进行clip处理，搜索对称的clip因子 $\argmin_{\Delta} \| Q(W\Delta)\Delta^{-1}X - WX \|_2^2$, grid search for $\alpha = \beta \in [0, 1]$
- **缺点**: AWQ算法通过**统一截断因子**进行权重截断，但在**非对称量化**时会影响低比特精度。
- **改进**: 为此，我们在**非对称量化**时搜索**不同截断因子**，而**对称量化**时保持**统一因子**，确保clip和量化过程**对齐**，实现更好的结果

#Bits	Method	LLaMA-2-7B		LLaMA-2-70B	
		Avg. PPL↓	Avg. Acc.↑	Avg. PPL↓	Avg. Acc.↑
w3a16g128	AWQ	7.25	61.18	4.90	80.95
	AWQ w/ asym. clip	7.21	61.59	4.89	81.07
w2a16g64	AWQ	1.8e5	37.69	6.8e4	32.84
	AWQ w/ asym. clip	13.26	48.77	6.49	75.31

- 有关于AWQ算法的使用配置文件可参考：
<https://github.com/ModelTC/llmc/tree/main/configs/quantization/methods/Awq>

OmniQuant

➤ 思想：学习等价变换的scaling、shift参数，以及weight量化的clip参数



Learnable weight clipping:

$$\mathbf{W}_q = \text{clamp}(\lfloor \frac{\mathbf{W}}{h} \rfloor + z, 0, 2^N - 1)$$

where $h = \frac{\gamma \max(\mathbf{W}) - \beta \min(\mathbf{W})}{2^N - 1}, z = -\lfloor \frac{\beta \min(\mathbf{W})}{h} \rfloor$

Learnable equivalent transformation:

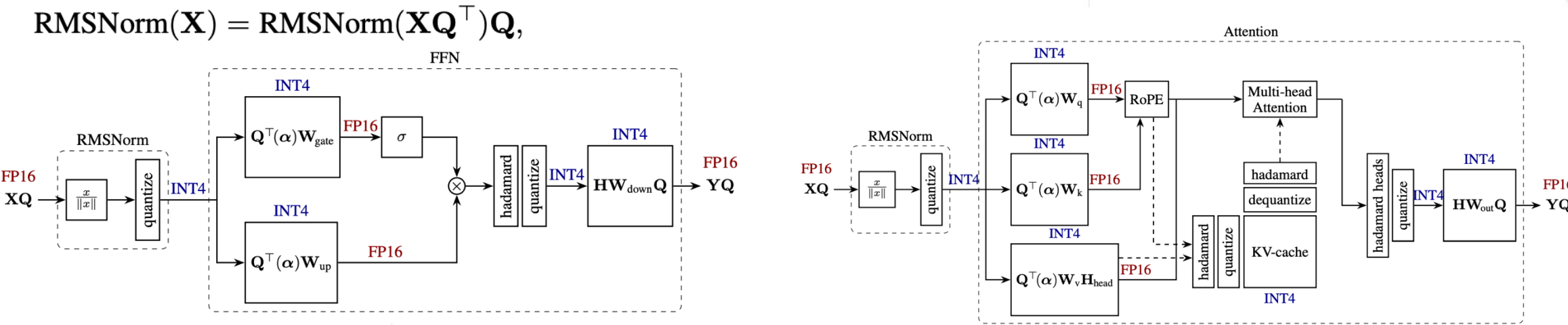
$$\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{B} = \underbrace{[(\mathbf{X} - \delta) \oslash s]}_{\tilde{\mathbf{X}}} \cdot \underbrace{[s \odot \mathbf{W}]}_{\tilde{\mathbf{W}}} + \underbrace{[\mathbf{B} + \delta \mathbf{W}]}_{\tilde{\mathbf{B}}}$$

- 缺点：我们发现OmniQuant的学习过程不稳定，且对超参数和初始化敏感。
- 改进：为此，我们改进了该过程，通过使用AWQ生成的scale和clip参数来初始化OmniQuant，从而显著减少了训练时间并提升了精度表现。配置文件可参考https://github.com/ModelTC/llmc/tree/main/configs/quantization/combination/awq_comb_omni/w2a16g64

LLaMA-2-7B-w2a16g64	wikitext2	c4	Epochs
OmniQuant	9.62	12.72	40
AWQ+OmniQuant	8.66	12.30	5

Quarot

➤ 思想：使用旋转矩阵Q，作用在weight和activation上，减少outlier

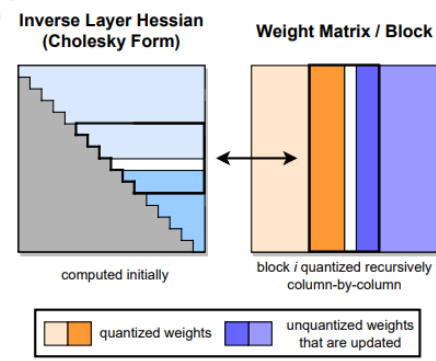


➤ 缺点：但其缺点在于Q是一个随机Hadamard矩阵，具有随机性。在旋转过程中没有考虑量化输出，可能导致尽管权重和激活的量化误差较小，但最终输出的量化误差较大。

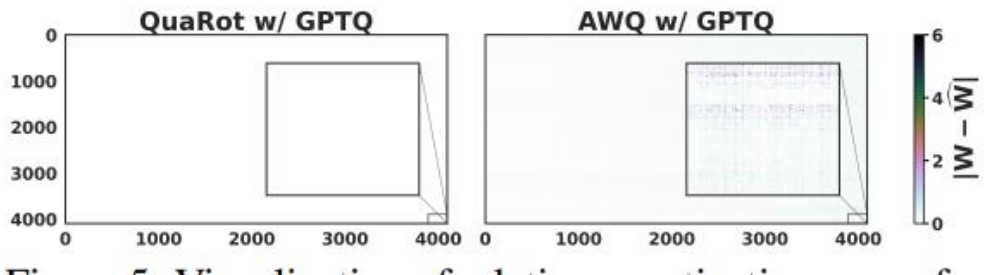
Method	q_proj	k_proj	v_proj	o_proj	gate_proj	up_proj	down_proj	PPL↓
Full Prec.	3.6505	4.3354	3.4174	3.4720	3.2991	3.2300	3.5845	6.14
AWQ	4.9219	6.1633	3.4602	3.4720	3.3190	3.2438	4.3083	8.57
	0.9960	0.9960	0.9784	0.9387	0.9882	0.9628	0.9479	
QuaRot	2.9051	2.9050	2.9069	2.9075	2.9074	2.9073	2.9075	40.81
	0.9962	0.9967	0.9797	0.8286	0.9764	0.9579	0.9230	

GPTQ

- **思想**: 在量化过程中, 针对某个块 (通常是若干列) 内的所有参数进行逐个量化。量化后, 需要适当调整该块内其他未量化的参数, 以弥补因量化导致的精度损失。



- **缺点**: 量化误差可能会累积到最后的某些列上, 导致这些通道的量化精度变差。
- **改进**: 结合Quarot和GPTQ的方法, 二者互补各自的缺点。GPTQ通过重建输出并调整权重的过程, 弥补了Quarot旋转矩阵随机性带来的问题。同时, Quarot后的权重矩阵中的离群值 (outlier) 现象得到了缓解, 从而减少了每个块的量化误差, 避免了最终一些参数因误差累积而导致的精度下降。配置文件可参考https://github.com/ModelTC/llmc/tree/main/configs/quantization/combination/quarot_comb_gptq/w8a8



Metric	GPTQ	AWQ	AWQ w/ GPTQ	QuaRot	QuaRot w/ GPTQ
Avg. PPL↓	10.67	10.98	10.55	50.00	10.35
Avg. Acc.↑	71.96	70.72	72.72	45.90	74.84

- 推理后端是实现量化加速的关键。LLMC能够根据不同推理后端所需的量化格式（如W8A8、W4A16、FP8等）导出模型，同时保证精度，确保在量化加速的同时不显著影响模型的性能。
- 以vLLM和Deepseekv2-Chat-Lite为例：

BF16

```
Loading safetensors checkpoint shards: 0% Completed | 0/4 [00:00<?, ?it/s]
Loading safetensors checkpoint shards: 25% Completed | 1/4 [00:01<00:04, 1.44s/it]
Loading safetensors checkpoint shards: 50% Completed | 2/4 [00:02<00:02, 1.22s/it]
Loading safetensors checkpoint shards: 75% Completed | 3/4 [00:03<00:01, 1.32s/it]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:05<00:00, 1.38s/it]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:05<00:00, 1.35s/it]

INFO 12-13 17:07:08 model_runner.py:1025] Loading model weights took 29.3010 GB
```

prompt: What is the AI?

generate: Artificial Intelligence (AI) is a branch of computer science that emphasizes the development of computer systems able to carry out tasks that would normally require human intelligence. These tasks include visual perception, speech recognition, decision-making, and language translation.

LLMC_FP8(E4M3)

```
WARNING 12-13 19:08:33 fp8.py:47] Detected fp8 checkpoint. Please note that the format is
Cache shape torch.Size([163840, 64])
Loading safetensors checkpoint shards: 0% Completed | 0/4 [00:00<?, ?it/s]
Loading safetensors checkpoint shards: 25% Completed | 1/4 [00:01<00:03, 1.00s/it]
Loading safetensors checkpoint shards: 50% Completed | 2/4 [00:01<00:01, 1.77it/s]
Loading safetensors checkpoint shards: 75% Completed | 3/4 [00:02<00:00, 1.31it/s]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:03<00:00, 1.16it/s]
Loading safetensors checkpoint shards: 100% Completed | 4/4 [00:03<00:00, 1.22it/s]

WARNING 12-13 19:08:36 utils.py:747] Found input scales that are not equal for fp8 MoE l
INFO 12-13 19:08:37 model_runner.py:1025] Loading model weights took 15.0753 GB
```

generate: Artificial Intelligence (AI) is a branch of computer science that emphasizes the development of computer systems able to carry out tasks that would normally require human intelligence. These tasks include learning, reasoning, problem-solving, perception, and language understanding.

现场演示 (下载代码/使用docker环境/修改config/修改bash/kill进程)

如何选择你的量化方案？

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4
- more things 1：输入是否有很关键的信息(wa/w only)

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4
- more things 1: 输入是否有很关键的信息(wa/w only)
- more things 2: prompt cache要考虑

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4
- more things 1: 输入是否有很关键的信息(wa/w only)
- more things 2: prompt cache要考虑
- more things 3: prompt cache预存

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4
- more things 1: 输入是否有很关键的信息(wa/w only)
- more things 2: prompt cache要考虑
- more things 3: prompt cache预存
- more things 4: 究竟快不快？不同model size? fp6? 非nv硬件？

如何选择你的量化方案？

- 寻找一个合适的推理后端
- 从推理后端支持的量化方案中选择
- 确定所支持的量化方案的细节：比特数，对称/非对称，均匀非均匀等。。。
- 确定你的硬件细节：fp8/int8，8bit/4bit，kv
- 根据你的业务模型特点选择：长输入/短输入，长输出/短输出
- 根据你的业务模型的精度选择：容忍度排名kv > act > weight。极端：kv4,a8,w4
- more things 1: 输入是否有很关键的信息(wa/w only)
- more things 2: prompt cache要考虑
- more things 3: prompt cache预存
- more things 4: 究竟快不快？不同model size? fp6? 非nv硬件?
- more things 5: fp8 is all you need.

如何选择你的量化算法？

- 使用良好的calib data

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行
- ppl的价值，自定义的数据集评测 (ppl/gen)

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行
- ppl的价值，自定义的数据集评测 (ppl/gen)
- training-base的方法真的好吗？

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行
- ppl的价值，自定义的数据集评测 (ppl/gen)
- training-base的方法真的好吗？
- prompt cache预存？

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行
- ppl的价值，自定义的数据集评测 (ppl/gen)
- training-base的方法真的好吗？
- prompt cache预存？
- 后端导出和新后端的接入

如何选择你的量化算法？

- 使用良好的calib data
- quarot + gptq
- 关闭在线旋转
- awq for down layer
- more data -> more mem -> 多卡并行
- ppl的价值，自定义的数据集评测 (ppl/gen)
- training-base的方法真的好吗？
- prompt cache预存？
- 后端导出和新后端的接入
- 多模态 (calib数据)

如何增加新模型？

- 找到模型推理逻辑，定义模型的subset
- 重写子类的一些基本成员函数
- 多模态模型相对复杂一些
 - 需要额外对齐数据处理逻辑
 - 多个模型部分的定义
 - 旋转等价性考虑

如何增加新算法？

- 大部分算法都是block wise优化
- 找到对应的优化逻辑，重写subset_transform

- 量化算子测速的工具: https://github.com/ModelTC/quant_horizon (我们将持续更新)

```
warnings.warn(
```

Kernel	Latency/s	Speed-Up
torch_linear	0.000111618	1
marlin_quant + perchannel	3.19028e-05	3.5
marlin_quant + pergroup_g128	3.51477e-05	3.18
marlin_quant_sparse + perchannel	2.73561e-05	4.08
marlin_quant_sparse + pergroup_g128	2.4581e-05	4.54
deepseed_quant + fp6_perchannel	5.39017e-05	2.07

```
2024-12-11 09:29:58.469 | ERROR | utils.registry_factory:benchm
```

Kernel	max_diff	cosine
torch_linear	0	1
marlin_quant + perchannel	52.8438	0.988427
marlin_quant + pergroup_g128	41.4062	0.993928
marlin_quant_sparse + perchannel	261.5	0.685653
marlin_quant_sparse + pergroup_g128	253.5	0.689946
deepseed_quant + fp6_perchannel	20.5625	0.998609

- 剪枝/稀疏现状
- 蒸馏现状
- 可以给llmc一些star, 以及希望大家也能参与到llmc/quant_horizon的建设中
- 教大家快速找到llmc在哪



坚持原创，让 AI 引领人类进步