大模型 – 容量估算

作者：Calvin
QQ: 179209347
Mail: 179209347@qq.com

介绍

笔记简介：

- 面向对象：深度学习初学者
- 依赖课程：**线性代数，统计概率**，优化理论，图论，离散数学，微积分，信息论

知乎专栏：

https://zhuanlan.zhihu.com/p/693738275

Github & Gitee 地址：

https://github.com/mymagicpower/AIAS/tree/main/deep_learning

https://gitee.com/mymagicpower/AIAS/tree/main/deep_learning

*** 版权声明：**

- 仅限用于个人学习
- 禁止用于任何商业用途

目录

CONTENTS

背景

LLM之推理成本高主要原因在于：

- 模型自身复杂性：模型参数规模大，对计算和内存的需求增加
- 自回归解码：逐token进行，速度太慢

推理加速一般可以从两个层面体现：延迟与吞吐量

- **延迟 (Latency)：** 主要从用户的视角来看，从提交一个prompt，至返回response的响应时间，衡量指标为生成单个token的速度，如16 ms/token；若batch_size=1，只给一个用户进行服务，Latency最低。
- **吞吐量 (Throughput)：** 主要从系统的角度来看，单位时间内能处理的token数，如16 tokens/s；增加Throughput一般通过增加batch_size，即，将多个用户的请求由串行改为并行

LLM 服务的四个关键指标

- **首次令牌时间 (TTFT)**：

用户输入查询后开始看到模型输出的速度。较短的响应等待时间对于实时交互至关重要，但对于离线工作负载则不太重要。该指标由处理提示然后生成第一个输出令牌所需的时间驱动。

- **每个输出令牌的时间 (TPOT)**：

为查询我们系统的每个用户生成输出令牌的时间。该指标与每个用户如何感知模型的“速度”相对应。例如，100 毫秒/token 的 TPOT 意味着每个用户每秒 10 个 token，或每分钟约 450 个单词，这比一般人的阅读速度要快。

- **延迟**：

模型为用户生成完整响应所需的总时间。总体响应延迟可以使用前两个指标来计算：
延迟 = (TTFT) + (TPOT) *（要生成的令牌数量）。

- **吞吐量**：

推理服务器每秒可以针对所有用户和请求生成的输出令牌数。

推理加速

延迟 (Latency)：
- 关注服务体验，返回结果越快，用户体验越好；
- 针对延迟的优化，主要还是底层的OP算子、矩阵优化、并行、更高效的C++推理等。针对延迟的优化可以提升吞吐量，但没有直接增加 batch_size 来的更显著。

吞吐量 (Throughput)：
- 关注系统成本，系统单位时间处理的量越大，系统利用率越高。
- 针对吞吐量优化，主要是KV Cache存取优化，本质是降低显存开销，从而可以增加 batch_size。

权衡 (trade-off)：
- 高并发时，将用户请求组batch后能提升吞吐量，但一定程度上会损害每个用户的延迟，因为之前只计算一个请求，现在合并计算多个请求，每个用户等待的时间就长了。通常，吞吐量随batch_size增大而增大，延迟也随之提升，当然延迟在可接受范围内就是ok的，因此二者需要trade-off。
- 对于一些离线的场景，我们对延迟并不敏感，这时会更多的关注吞吐量，但对于一些实时交互类的应用，我们就需要同时考虑延迟和吞吐量。

推理加速

权衡 (trade-off)：

- 高并发时，将用户请求组batch后能提升吞吐量，但一定程度上会损害每个用户的延迟，因为之前只计算一个请求，现在合并计算多个请求，每个用户等待的时间就长了。在达到一定的批量大小之后，即当我们进入计算限制状态时，批量大小的每次加倍只会增加延迟，而不会增加吞吐量。通常，吞吐量随batch_size增大而增大，延迟也随之提升，当然延迟在可接受范围内就是ok的，因此二者需要trade-off。
- 对于一些离线的场景，我们对延迟并不敏感，这时会更多的关注吞吐量，但对于一些实时交互类的应用，我们就需要同时考虑延迟和吞吐量。

BS	输入	输出	时延	QPS
1	128	8	353.33ms	2.83
2	128	8	386.81ms	5.17
4	128	8	449.59ms	8.89
8	128	8	592.87ms	13.49
16	128	8	833.05ms	19.21
			测试环境：8卡A100(80G)	

以 Batch size 为 16 时为例，生成 8 个 token 所需的时间为 0.833 秒

数据来源：百度智能云

Transformer 结构优化

该类方法主要通过优化 Transformer 的结构以实现推理性能的提升。

在自回归 decoder 中，所有输入到 LLM 的 token 会产生注意力 key 和 value 的张量，这些张量保存在 GPU 显存中以生成下一个 token。这些缓存 key 和 value 的张量通常被称为 KV cache，其具有以下特点：

- 显存占用大：在 LLaMA-13B 中，缓存单个序列最多需要 1.7GB 显存；
- 动态变化：KV 缓存的大小取决于序列长度，这是高度可变和不可预测的。因此，这对有效管理 KV cache 挑战较大。该研究发现，由于碎片化和过度保留，现有系统浪费了 60% - 80% 的显存。

4.1 FlashAttention

- 在不访问整个输入的情况下计算 softmax
- 不为反向传播存储大的中间 attention 矩阵

4.2 PagedAttention

PagedAttention 允许在非连续的内存空间中存储连续的 key 和 value 。
具体来说，PagedAttention 将每个序列的 KV cache 划分为块，每个块包含固定数量 token 的键和值。在注意力计算期间，PagedAttention 内核可以有效地识别和获取这些块。

4.3 FLAT Attention

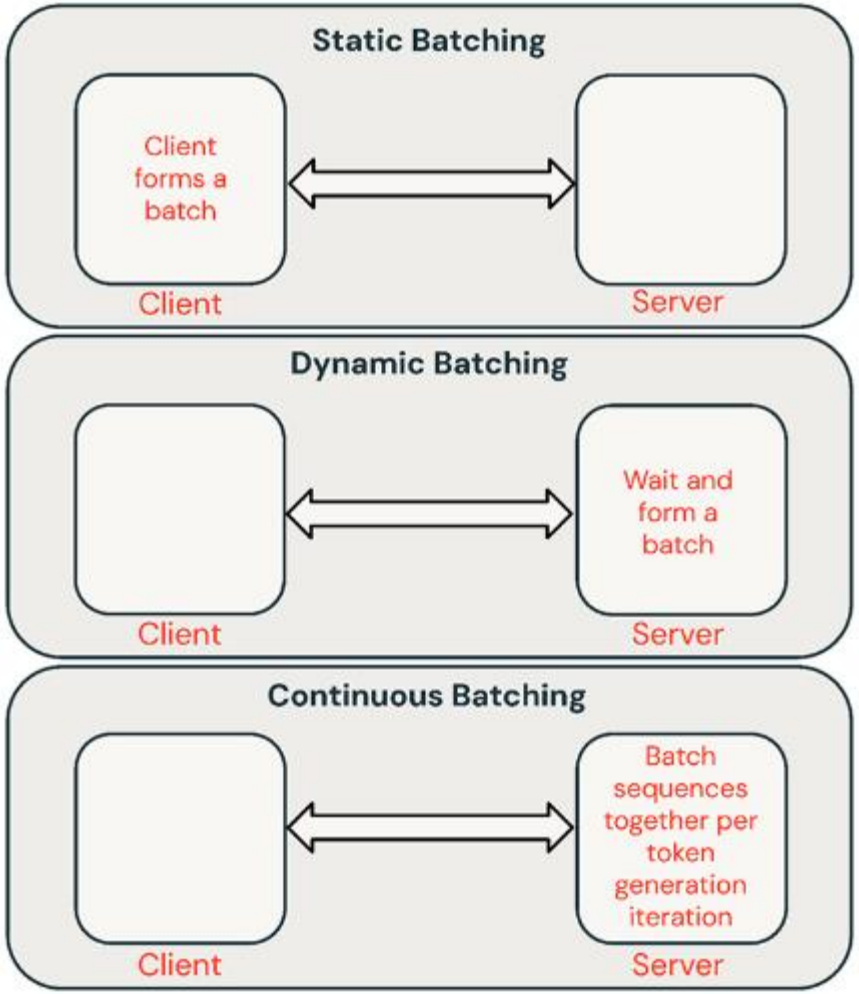
FLAT-Attention 与 FlashAttention 采取不同的路线来解决同一问题。 提出的解决方案有所不同，但关键思想是相同的（tiling 和 scheudling）。

吞吐量

我们可以通过将请求一起批处理来权衡每个token的吞吐量和时间。与按顺序处理查询相比，在 GPU 评估期间对查询进行批量推理可提高吞吐量，但每个查询将需要更长的时间才能完成（忽略排队效应）。

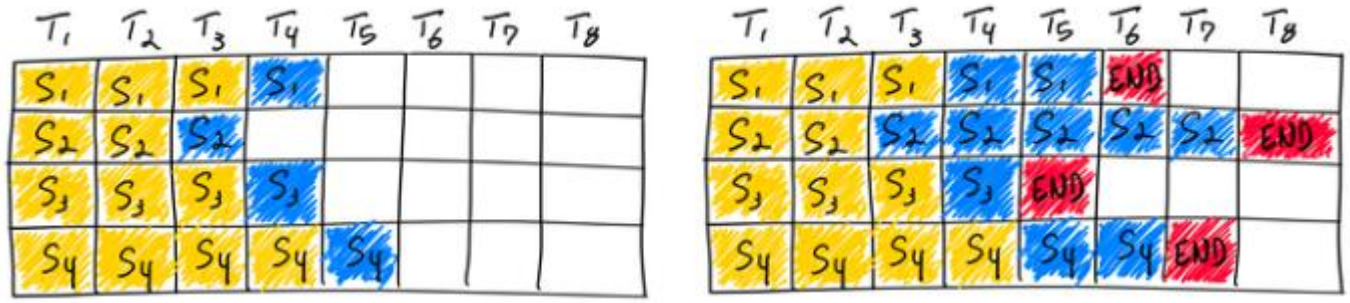
有一些用于批量推理请求的常用技术：
静态批处理：客户端将多个提示打包到请求中，并在批处理中的所有序列完成后返回响应。我们的推理服务器支持这一点，但不要求它。
动态批处理：在服务器内动态批处理在一起。通常，此方法的性能比静态批处理差，但如果响应较短或长度统一，则可以接近最优。当请求具有不同参数时效果不佳。

连续批处理：目前是 SOTA 方法。它不是等待批次中的所有序列完成，而是在迭代级别将序列分组在一起。它可以实现比动态批处理高 10 倍到 20 倍的吞吐量。

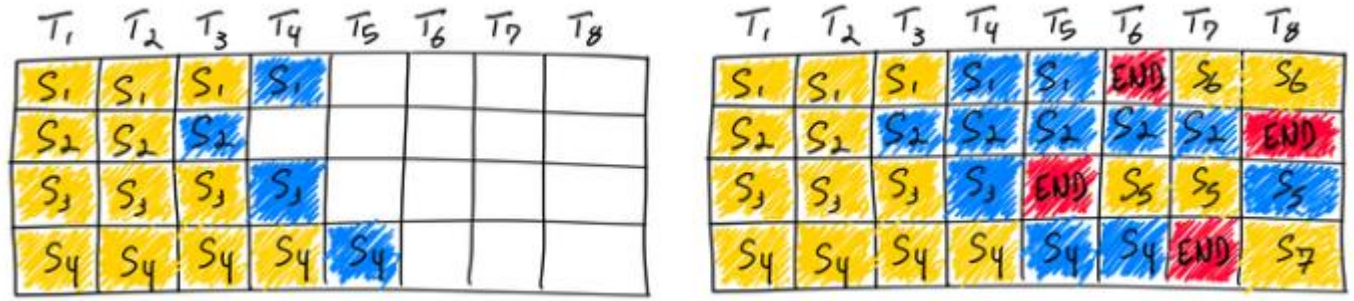


连续批处理（Continuous batch）

该类方法主要是针对多 Batch 的场景，通过对 Batch 的时序优化，以达到去除 padding、提高吞吐和设备利用率。传统的 Batch 处理方法是静态的，因为Batch size 的大小在推理完成之前保持不变。



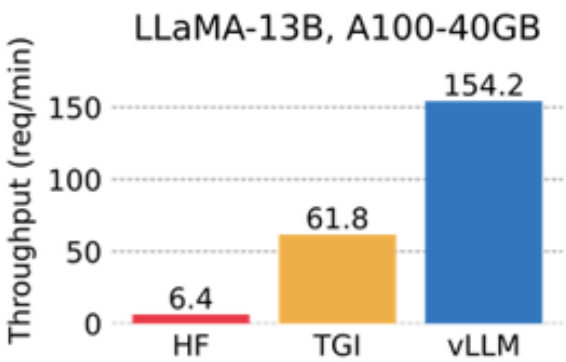
Continuous Batch不是等到 Batch 中的所有序列完成生成，而是实现 iteration 级调度，其中Batch size 由每次迭代确定。 结果是，一旦 Batch 中的序列完成生成，就可以在其位置插入新序列，从而比静态 Batch 产生更高的 GPU 利用率。



vLLM

https://vllm.ai/

吞吐性能



功能:
- 支持动态 batch。
- 使用 **PagedAttention** 优化 KV cache 显存管理。
- 支持linux。

优点:
- 最快的推理速度
- 最好的服务吞吐性能
- 与 HuggingFace 模型无缝集成（目前支持GPT2, GPTNeo, LLaMA, OPT 系列）
- 高吞吐量服务与各种 decoder 算法，包括并行采样、beam search 等
- 张量并行(TP)以支持分布式推理
- 流输出
- 兼容 OpenAI 的 API 服务

缺点:
- 添加自定义模型：虽然可以合并自己的模型，但如果模型没有使用与vLLM中现有模型类似的架构，则过程会变得更加复杂。例如，增加Falcon的支持，这似乎很有挑战性；
- 缺乏对适配器（LoRA、QLoRA等）的支持：当针对特定任务进行微调时，开源LLM具有重要价值。然而，在当前的实现中，没有单独使用模型和适配器权重的选项，这限制了有效利用此类模型的灵活性。
- 缺少权重量化：这对于减少GPU显存消耗至关重要。

vLLM支持的模型

- Aquila & Aquila2 (BAAI/AquilaChat2-7B, BAAI/AquilaChat2-34B, BAAI/Aquila-7B, BAAI/AquilaChat-7B, etc.)
- Baichuan (baichuan-inc/Baichuan-7B, baichuan-inc/Baichuan-13B-Chat, etc.)
- BLOOM (bigscience/bloom, bigscience/bloomz, etc.)
- ChatGLM (THUDM/chatglm2-6b, THUDM/chatglm3-6b, etc.)
- Falcon (tiiuae/falcon-7b, tiiuae/falcon-40b, tiiuae/falcon-rw-7b, etc.)
- GPT-2 (gpt2, gpt2-xl, etc.)
- GPT BigCode (bigcode/starcoder, bigcode/gpt_bigcode-santacoder, etc.)
- GPT-J (EleutherAI/gpt-j-6b, nomic-ai/gpt4all-j, etc.)
- GPT-NeoX (EleutherAI/gpt-neox-20b, databricks/dolly-v2-12b, stabilityai/stablelm-tuned-alpha-7b, etc.)
- InternLM (internlm/internlm-7b, internlm/internlm-chat-7b, etc.)
- LLaMA & LLaMA-2 (meta-llama/Llama-2-70b-hf, lmsys/vicuna-13b-v1.3, young-geng/koala, openlm-research/open_llama_13b, etc.)
- Mistral (mistralai/Mistral-7B-v0.1, mistralai/Mistral-7B-Instruct-v0.1, etc.)
- MPT (mosaicml/mpt-7b, mosaicml/mpt-30b, etc.)
- OPT (facebook/opt-66b, facebook/opt-iml-max-30b, etc.)
- Phi-1.5 (microsoft/phi-1_5, etc.)
- Qwen (Qwen/Qwen-7B, Qwen/Qwen-7B-Chat, etc.)
- Yi (01-ai/Yi-6B, 01-ai/Yi-34B, etc.)

https://github.com/vllm-project/vllm

目录

CONTENTS

推理显存计算

模型推理（inference）是指在已经训练好的模型上对新的数据进行预测或分类。推理阶段通常比训练阶段要求更低的显存，因为不涉及梯度计算和参数更新等大量计算。以下是计算模型推理时所需显存的一些关键因素：

模型结构： 模型的结构包括层数、每层的神经元数量、卷积核大小等。较深的模型通常需要更多的显存，因为每一层都会产生中间计算结果。

输入数据： 推理时所需的显存与输入数据的尺寸有关。更大尺寸的输入数据会占用更多的显存。

批处理大小 BatchSize： 批处理大小是指一次推理中处理的样本数量。较大的批处理大小可能会增加显存使用，因为需要同时存储多个样本的计算结果。

数据类型： 使用的数据类型（如单精度浮点数、半精度浮点数）也会影响显存需求。较低精度的数据类型通常会减少显存需求。

中间计算： 在模型的推理过程中，可能会产生一些中间计算结果，这些中间结果也会占用一定的显存。

A10 显卡

FP16张量核心：这是计算带宽。在半精度（也称为FP16）模型中拥有125 TFLOPS（每秒兆浮点运算次数）。半精度每个数占用16位。

GPU显存：我们可以通过将参数数量（以B为单位）乘以2来快速估算模型的大小（以G为单位）。这种方法基于一个简单的公式：每个参数在半精度中使用16位（或2字节）的显存，显存使用量大约是参数数量的两倍。
因此，例如，一个70亿参数的模型大约需要占用14 GB的显存。A10 GPU的24 GB VRAM，还剩下约10 GB的显存作为缓冲区。

GPU显存带宽：我们可以从GPU显存（也称为HBM或高带宽显存）向芯片上的处理单元（也称为SRAM或共享内存）传输600 GB/s。

A 列	B 列
FP32	31.2 TF
TF32 Tensor Core	62.5 TF | 125 TF*
BFLOAT16 Tensor Core	125 TF | 250 TF*
FP16 Tensor Core	125 TF | 250 TF*
INT8 Tensor Core	250 TOPS | 500 TOPS*
INT4 Tensor Core	500 TOPS | 1000 TOPS*
GPU Memory	24 GB GDDR6
GPU Memory Bandwidth	600 GB/s
Max TDP Power	150W

计算访存比

访存比：每访问一字节显存，我们可以完成多少浮点运算每秒（FLOPS）。
根据规格表中的数字，我们计算A10的访存比：

非稀疏矩阵计算 ops_to_byte_A10
= compute_bw / memory_bw
= 125 TF / 600 GB/S
= 208.3 ops / byte

稀疏性矩阵计算 ops_to_byte_A10
= compute_bw / memory_bw
= 250 TF / 600 GB/S
= 416.7 ops / byte

以非稀疏矩阵计算为例：

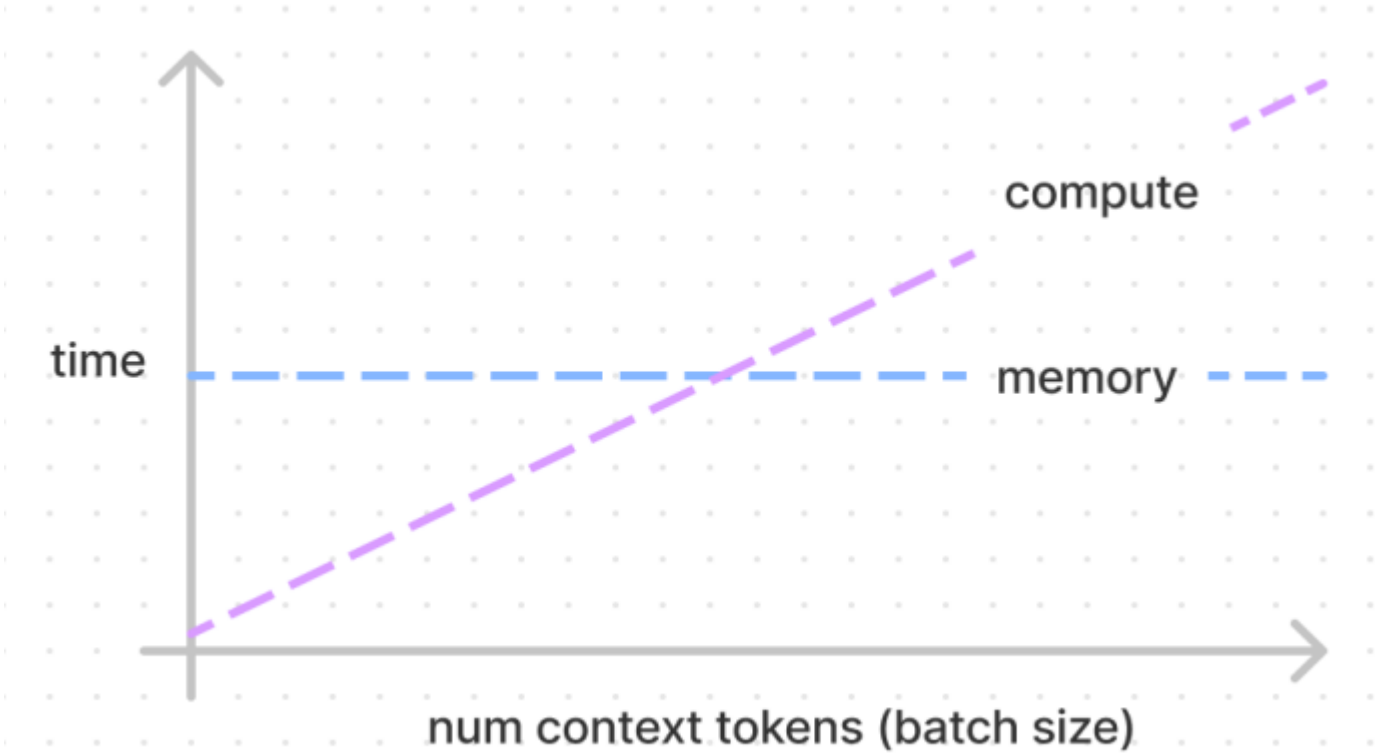
上面的数字意味着为了充分利用我们的计算资源，我们需要完成每个显存访问字节的208.3个浮点运算。

- 如果我们发现每个字节只能完成少于208.3个运算，那么我们的系统性能**受限于显存吞吐**。这基本上意味着我们的系统速度和效率受限于数据传输速率或其所能处理的输入输出操作的速度。
- 如果我们希望每个字节执行超过208.3个浮点运算，那么我们的系统**受限于计算能力**。在这种情况下，我们的效率和性能受限于芯片所拥有的计算单元数量，而不是显存。

了解我们是受限于计算能力还是显存是至关重要的，这样我们就知道在哪个方面应该集中优化工作。

计算访存比

非稀疏矩阵计算为例：下图的交点是208，不过实际上内存线（memory line）会有一些倾斜，这是因为中间计算（intermediate calculation）存在显存成本。



这意味着如果我们要计算一个token的kv，那么计算多达208个token的kv所需的时间将是相同的！

- 如果低于这个数量，我们将受到显存带宽的限制；
- 如果超过这个数量，我们将受到FLOPS的限制。

计算访存比总结

AI 芯片的计算速度“远快于”显存带宽。从英伟达各类芯片本身 的“计算访存能力比”来看，对于 Transformer 模型生成这类访存密集型任务，决 定生成速度的不是芯片的强项 FLOPS 能力，而是显存的带宽。我们认为，针对大 模型推理这类访存密集型任务，对其算力需求的估计，不能单单考虑其 FLOPs 的 需求，更重要的瓶颈在于显存带宽。

	T4	V100	A10		A100 SXM		H800SXM	
显存带宽(GB/s)	320	1134	600		2039		3350	
Fp16 (TFLOPS)	65	130	125	250	312	624(稀疏)	990	1979(稀疏)
“计算访存比” (fp16 算力/显存带宽)	203	114.6	208.3	416.7	153	306	295.5	590.7

TFLOPS=10^12 FLOPS，GB/s=10^9 B/s

目录

CONTENTS

每个token进行一次前向计算所需算力公式

Operation	Parameters	FLOPs per Token
Embed	$(n_{\text{vocab}} + n_{\text{ctx}})\, d_{\text{model}}$	$4d_{\text{model}}$
Attention: QKV	$n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$	$2n_{\text{layer}} d_{\text{model}} 3d_{\text{attn}}$
Attention: Mask	—	$2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$
Attention: Project	$n_{\text{layer}} d_{\text{attn}} d_{\text{model}}$	$2n_{\text{layer}} d_{\text{attn}} d_{\text{embd}}$
Feedforward	$n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$	$2n_{\text{layer}} 2d_{\text{model}} d_{\text{ff}}$
De-embed	—	$2d_{\text{model}} n_{\text{vocab}}$
Total (Non-Embedding)	$N = 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}})$	$C_{\text{forward}} = 2N + 2n_{\text{layer}} n_{\text{ctx}} d_{\text{attn}}$

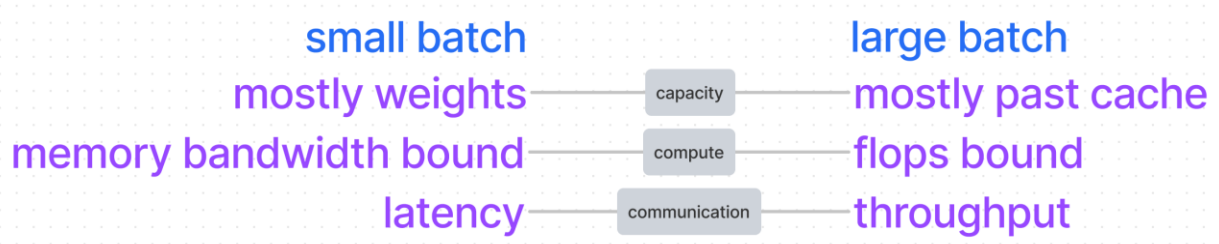
- d_{ff}, dimension of the intermediate feed-forward layer
- d_{attn} , dimension of the attention output, d_{attn} = d_{ff} /4 = d_{model}
- n_{heads}, number of attention heads per layer
- n_{layers} , number of layers
- n_{ctx} , tokens in the input context
- d_{model} , dimension of the residual stream

总参数量 $P = N \approx 2d_{\text{model}} n_{\text{layer}} (2d_{\text{attn}} + d_{\text{ff}}) = 12 n_{\text{layer}} d_{\text{model}}^2$ 总算力

flops运算次数 $C \approx 2P$

https://arxiv.org/pdf/2001.08361.pdf

公式总结



总参数量 P

$$12 n_{\text{layer}} d_{\text{model}}^2$$

flops运算次数

$$2\ P$$

模型大小（FP 16 每个参数都是2字节）

$$2\ P$$

计算单个token解码步骤的时延时间需要两个公式：一个用于显存带宽bound（小batch），另一个用于 flops bound（大batch）。

小batch
（如：batch size=1）

单token访存带宽时间

$$\frac{2 \cdot P}{N \cdot A_{\text{bm}}}$$

通讯时间

$$4 \cdot n_{\text{layers}} \cdot 8\mu s$$

其中：N为GPU数量（显存带宽在多个GPU间分担），A_{bm}为GPU显存带宽

大 batch
（如：batch size=512）

单token计算时间

$$B \cdot \frac{2 \cdot P}{N \cdot A_f}$$

通讯时间

$$B \cdot \frac{2 \cdot 4 \cdot n_{\text{layers}} \cdot d_{\text{model}}}{A_c}$$

其中：B 是批量大小，A_c为NVLink通讯带宽

时延计算

在16个GPU上使用260B参数的Gopher模型来进行推理。使用小batch推理时，生成一个token需要22毫秒的时间。通过大batch公式计算出来的通信吞吐量成本约为35微秒，因此可以忽略。

小batch
(如：batch size=1)

单token访存带宽时间

$$\frac{2 \cdot P}{N \cdot A_{\text{bm}}} = \frac{2 \cdot 260\text{e}9}{16 \cdot 1.5\text{e}12} \approx 0.0217 \approx 22\text{ms}$$

单token计算时间

$$B \cdot \frac{2 \cdot P}{N \cdot A_f} = 1 \cdot \frac{2 \cdot 260\text{e}9}{16 \cdot 312\text{e}12} \approx 104\mu s$$

通讯时间（NVLink）

$$2 \cdot 4 \cdot n_{\text{layers}} \cdot 8\mu s = 4 \cdot 80 \cdot 8\mu s = 2560\mu s \approx 3\text{ms}$$

通讯时间

$$B \cdot \frac{2 \cdot 4 \cdot n_{\text{layers}} \cdot d_{\text{model}}}{A_c} = 1 \cdot \frac{8 \cdot 80 \cdot 16384}{300\text{e}9} \approx 35\mu s$$

https://kipp.ly/transformer-inference-arithmetic/

时延计算

对于512的大batch，生成每个token所需的时间为53毫秒（即在62毫秒内生成512个token）。通信的时延成本也为3毫秒（由于消息可以一起准备，因此延时延不会随着batch的增加而增加），这对于减少时延来说有一定的意义，但如果假设通信和计算是并行的，那么这也可以接受。

大 batch
(如：batch size=512)

单token计算时间

$$B \cdot \frac{2 \cdot P}{N \cdot A_f} = 512 \cdot \frac{2 \cdot 260e9}{16 \cdot 312e12} \approx 0.053 \approx 53ms$$

通讯时间

$$B \cdot \frac{2 \cdot 4 \cdot n_{\text{layers}} \cdot d_{\text{model}}}{A_c} = 512 \cdot \frac{8 \cdot 80 \cdot 16384}{300e9} \approx 18ms$$

在假定并行处理的情况下，我们会选择计算和通信中较大的值。因此，我们希望避免通信时间大于计算时间（这是防止随着芯片数量的增加趋近于零时延的机制，最终通信时间将占用越来越多的时间）。

但并不能保证所有系统都能够完美地并行处理。这些数值明显比实际应用环境中得到的值要低得多。在这个环境中，程序只能使用被授权的资源，无法对系统或其他程序产生影响），因为它假设了最佳的硬件使用，没有考虑softmax，假设没有通信时延，并忽略了许多其他较小的因素。

尽管如此，这些数学推理仍有助于思考如何优化性能以及未来优化所带来的变化。

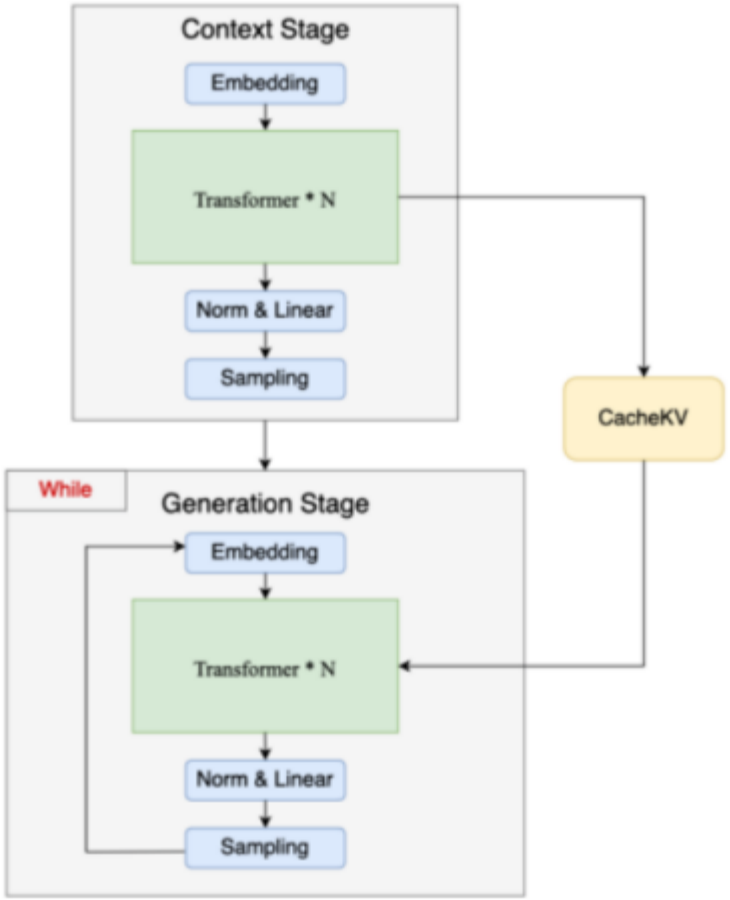
https://kipp.ly/transformer-inference-arithmetic/

目录

CONTENTS

GPT-3 为例生成阶段计算量与访存量需求

推理实际上涉及两个阶段，首先是将输入进行 Encode（Context 阶段），然后进入重复多次（while 循环）的 Generation 过程，两个阶段的所需计 算量与输入输出的大小与比值相关。



GPT-3 175B模型，输入1000 tokens，输出250 tokens

Context

BS	激活shape	计算量	访存量	计算访存比
1	[1, 1000, 12288]	2.743E+12	1.306E+11	21.00
4	[4, 1000, 12288]	1.097E+13	1.312E+11	83.61
8	[8, 1000, 12288]	2.194E+13	1.319E+11	166.33
16	[16, 1000, 12288]	4.388E+13	1.334E+11	328.94

特点：激活显存占用大，计算密集型

Generation

BS	激活shape	计算量	访存量	计算访存比
1	[1, 1, 12288]	6.836E+12	3.262E+13	0.20
4	[4, 1, 12288]	2.749E+13	3.264E+13	0.84
8	[8, 1, 12288]	5.537E+13	3.267E+13	1.67
16	[16, 1, 12288]	1.123E+14	3.273E+13	3.43

特点：激活显存占用小，访存密集型

数据来源：百度智能云

GPT-3.5 推理算力需求公式计算

模型端推理算力消耗					
A:输入 token 数	128	128	128	128	128
B:输出 token 数	8	8	8	8	8
C:BS	1	2	4	8	16
D:总输出 token 数 B*C	8	16	32	64	128
E:总处理 token 数 (A+B)*C	136	272	544	1088	2176
现行推理算力估算公式	**推理算力消耗=2*参数量*token 数**				
F:公式得出的算力消耗(TFLOPs)-仅计算输出 2*175B*D	• 2.8	• 5.6	• 11.2	• 22.4	44.8
G:公式得出的算力消耗(TFLOPs)-计算输入+输出 2*175B*E	• 47.6	• 95.2	• 190.4	• 380.8	761.6
芯片端算力供给					
• H:时延/用时(秒) – 实际测试数据	0.35	0.39	0.45	0.59	0.83
• I:A100 FP16 算力(TFLOPS)	624				
• J:8 卡合计(TFLOPS) 8*I	4992				
• K:8 卡对应时间内可提供算力(TFLOPs) J*H	1764	1931	2244	2960	4159
消耗/供给得出的："算力利用率"					
• 仅计算输出时"算力利用率" F/K	• 0.2%	• 0.3%	• 0.5%	• 0.8%	1.1%
• 计算输入+输出时"算力利用率" G/K	• 2.7%	• 4.9%	• 8.5%	• 12.9%	18.3%

数据来源：百度智能云，天翼智库，英伟达官网，财通证券研究所
注：该“算力利用率”仅为公式计算出的算力消耗与芯片可提供算力间比值，而非实际 GPU 运行时间角度的利用率

GPT-3.5 所需推理卡数量测算

指标	数值	备注
A: 模型参数量(亿个)	1750	展示以 GPT3.5 为例
B: 月均访问量(亿次)	16	全球网站访问榜月均：第 1 谷歌 850 亿 第 2 youtube 330 亿 第 3 facebook 178 亿 每次访问时长 10-20 分钟不等
C: 平均每次使用时间（分钟）	7.5	
D: 平均实时并发（万个）	27.8	[B*C/(24*60*30)]*10000
E: 平均每秒 tokens 输出（个）	20	假设 GPT3.5 平均每秒 20 个左右。
F: 峰值预留倍数	10	综合考虑并发峰值，以及显存预留等因素
G: "算力利用率"	15%	以显存带宽为代表的因素拖累算力利用率
H: 所需算力储备（FLOPS）	1.3E+20	2*A*(D*E*F)/G
假设精度	所需 A100(万块)	A100 算力
TF32	41.55	312 (TFLOPS=10^12FLOPS)　H/312 TFLOPS
TF16	20.78	624 (TFLOPS=10^12FLOPS)　H/624 TFLOPS
INT8	10.39	1248 (TFLOPS=10^12FLOPS)　H/1248 TFLOPS
INT4	5.19	2496 (TFLOPS=10^12FLOPS)　H/2496 TFLOPS

数据来源：Similarweb，英伟达官网，财通证券研究所

目录

CONTENTS

注意力(attention)计算过程

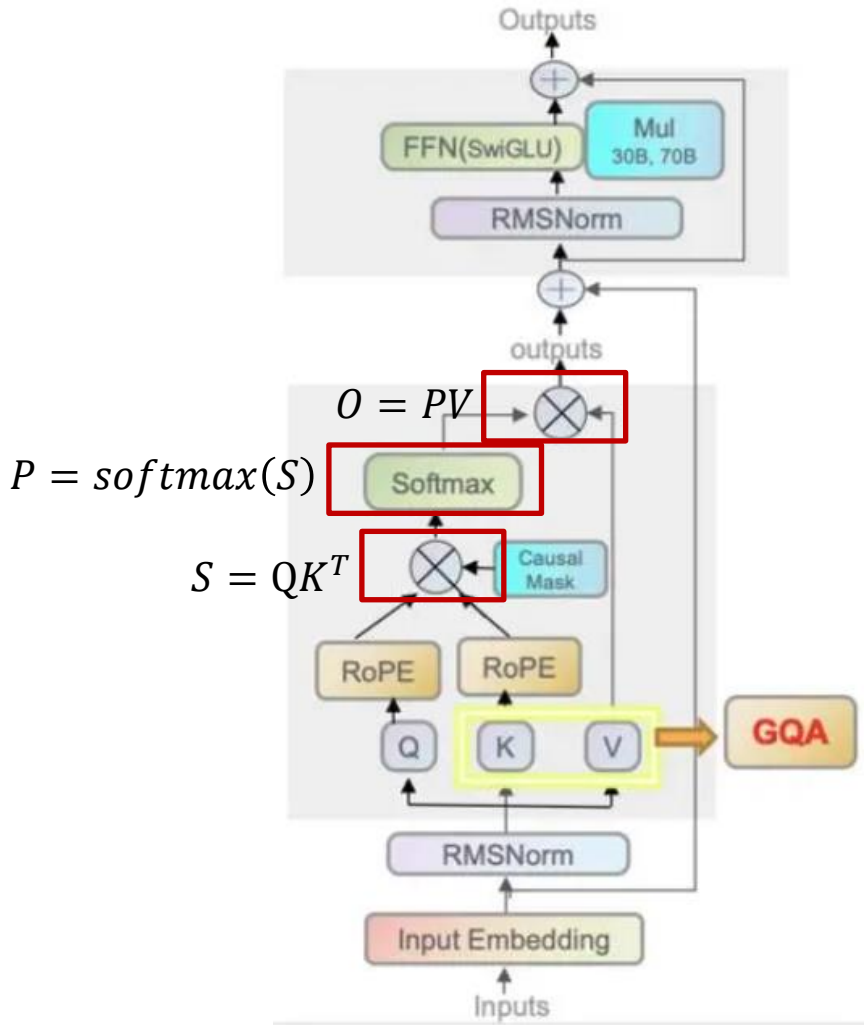
$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

(系数$\frac{1}{\sqrt{d_k}}$不影响复杂度计算，忽略)

Q，K，V $\in \mathbb{R}^{Nxd}$ 保存于 HBM (high bandwidth memory)：

1. 从HBM 读取 Q, K，计算 $S = QK^T$, 然后将 S 回写至 HBM
2. 从HBM 读取 S，计算 $P = softmax(S)$，然后将 P 回写至 HBM
3. 从HBM 读取 P, V，计算 $O = PV$, 然后将 O 回写至 HBM
4. 返回 O

每次执行 GPU 内核时，我们都需要将数据从 GPU 的 DRAM（HBM）移出或移回。



llama2 7B 架构

- d，可以表示为 d_{head}，是单个注意力头的维度。
  - 对于 7B d = 128
- n_{heads} 是注意力头的数量
  - 对于 7B n_{heads} = 32
- n_layers是注意力块出现的次数
  - 对于 7B n_{layers} = 32
- d_{model}，是模型的维度。 $d_{\text{model}} = n_{\text{heads}} \cdot d$
  - 对于 7B d_{model} = 4096

Llama 2的其他尺寸具有较大的 d_{model} （请参阅“dimension”列）。

params	dimension	n heads	n layers
6.7B	4096	32	32
13.0B	5120	40	40
32.5B	6656	52	60
65.2B	8192	64	80

https://arxiv.org/pdf/2302.13971.pdf

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

$$S = QK^T$$

$$P = softmax(S)\ (\text{系数}\frac{1}{\sqrt{d_k}}\text{不影响复杂度计算})$$

$$O = PV$$

KV 近似缓存计算

与token嵌入（token embeddings）相乘权重为 $W_k, W_v \in R^{d_{model} \times d_{model}}$
其中每个token嵌入为 $t_e \in R^{1 \times d_{model}}$，这样，我们就可以算出所有层的k和v需进行的浮点运算次数为：

$$2 \cdot 2 \cdot \text{n\_layers} \cdot d_{\text{model}}{}^2$$

将 t_e乘以W_k需要进行 $2 \cdot d_{\text{model}}{}^2$次浮点运算。另一个2表示我们需要重复两次这样的操作，一次用于计算k和一次用于计算v，然后再重复所有层数 n_layers。

矩阵乘法（matmul）中的浮点运算次数：

- 矩阵-向量（matrix-vector）乘法的计算公式是2mn，其中$A \in R^{m \times n}, b \in R^n$
- 矩阵-矩阵（matrix-matrix）乘法，计算公式是 2mnp，其中$A \in R^{m \times n}, B \in R^{n \times p}$

mn因子十分重要，因为它反映了矩阵乘法中由乘法和加法组成的组合方式，即“乘法(1)-加法(2) 操作组合”。

KV 近似缓存计算

这意味着对于一个520亿参数的模型来说 （以Anthropic中的模型为例，dmodel=8192， n_layers =64 ），其浮点运算次数为：

$$2 \cdot 2 \cdot 64 \cdot 8192^2 = 17{,}179{,}869{,}184$$

假设有一个A100 GPU，其每秒可执行的稀疏浮点运算次数为624e12，其显存带宽可达2e12字节/秒。以下数字仅涉及kv权重及计算的数值：

$$\text{memory} = 2 \cdot 2 \cdot n_{\text{layers}} \cdot {d_{\text{model}}}^2 \div 2\text{e}12$$

$$\text{compute} = 2 \cdot 2 \cdot n_{\text{layers}} \cdot {d_{\text{model}}}^2 \div 624\text{e}12$$

批处理

批处理通过在相同数量的显存加载和存储的情况下执行更多计算，增加了模型的计算密集度，从而减少了模型受显存限制的程度。

我们可以将批次大小设置为多大？在加载我们的70亿参数模型后，我们的A10上还剩下10 GB的显存：

```
24 GB - (2 * 7GB) = 10GB
```

llama2 KV 缓存

在半精度（FP16）下，每个浮点数需要2个字节来存储。有2个矩阵，为了计算KV缓存大小，我们将两者都乘以n_layers和d_model，得到以下方程：

每个token kv缓存 kv_cache_size：

= (2 * 2 * n_{layers} * d_{model})

= (2 * 2 * n_{layers} * n_{heads} * d)

= (4 * 32 * 4096)

= 524288 **bytes/token**

~ 0.00052 GB/token

kv_cache_tokens

= 10 GB / 0.00052 GB/token

= 19,230 tokens

我们的KV缓存可以轻松容纳19,230个 token。因此，对于Llama 2的标准序列长度4096个 token，我们的系统有足够的带宽来同时处理 4 个序列的批次。
总之，为了充分利用计算能力，我们希望在推断期间每次批处理4个请求，以填满 KV 缓存。这将增加我们的吞吐量。
如果使用LLM（语言模型）来异步处理大量文档队列，批处理是一个很好的主意。与逐个处理每个元素相比，将更快地处理队列，并且可以安排推理调用以快速填充批次，从而最大程度地减少对延迟的影响。

llama2 KV 缓存

第一个因子2表示k和v这两个向量。在每一层中我们都要存储这些k，v向量，在半精度（FP16）下，每个浮点数需要2个字节来存储。为了计算KV缓存大小，我们将两者都乘以 n_layers 和 d_model，得到以下方程：

每个token kv缓存 kv_cache_size：

$$= (2 * 2 * n_{\text{layers}} * d_{\text{model}})$$

$$= (2 * 2 * n_{\text{layers}} * n_{\text{heads}} * d)$$

KV Cache推理优化

kv缓存是为了避免每次采样token时重新计算键值向量。利用预先计算好的k值和v值，可以节省大量计算时间，尽管这会占用一定的存储空间。

减少头数 MQA/GQA

- 一种手段是减少KV heads的数量，如果以MQA(Multi-Query-Attention)来说，KV head 8->1 之间节省7倍存储量
- 对于GQA(Grouped-Query_Attention)来说，平衡精度将KV head 8 -> N, 1<N<8之间trading off精度和速度

减少Length长度

- 减少KV-Cache的长度， base窗口：Mistral-7B, StreamingLLM, LongLoRA
- 将Cache长度固定在sliding-window长度，借助RollingBuffer，不需要频繁“移位”
- 另外StreamingLLM采用torch.tensorsplit/cat操作实现
- 在lit-LLama中采用"rolling"算子管理cache的更新

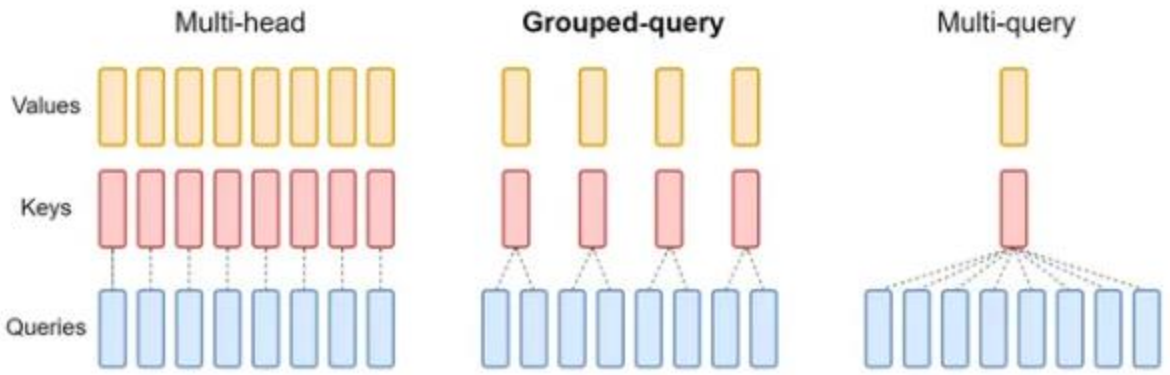
KV-Cache的管理，减少碎片

- 优化KV-Cache存储模型：PagedAttention
- 对于以下连续存储模型，如果将Page Length改成4
- PagedAttention不是去改变Attention的计算，而是改变KV-cache的存取方式

减少bits数，量化模型

LLM-QAT在量化训练过程，将KV-Cache也做Quantization
在部署时KV-Cache如果是16bit，量化成4bit的话，显存直接减少4倍

减少头数 MQA/GQA



公式
2 x seqlen x batch_size x [d x n_kv_heads] x n_layers x k-bits

公式
2 x seqlen x batch_size x D[d x n_kv_heads] x n_layers x k-bits

- n_kv_heads: MQA/GQA通过减少KV的头数减少显存占用
- seqlen : 通过减少长度 seqlen, 以减少KV显存占用，如使用循环队列管理窗口KV
- KV-Cache 的管理：从OS(操作系统)的内存管理角度，减少碎片，如Paged Attention
- K-bits: 从量化角度减少KV cache的宽度，如使用LLM-QAT进行量化

llama2 KV 缓存

Llama2 模型使用一种称为分组查询注意（GQA）的注意力变体。当 KV 头数为 1 时，GQA 与 Multi-Query-Attention (MQA) 相同。

GQA 通过共享键/值来帮助缩小 KV 缓存大小。KV缓存大小的计算公式为：
batch_size * seqlen * (d_model * n_kv_heads / n_heads) * n_layers * 2 (K and V) * 2 (bytes per Float16)

批量大小	GQA KV 高速缓存 (FP16)	GQA KV 缓存 (Int8)
1	0.312 GiB	0.156 GiB
16	5GiB	2.5GiB
32	10GiB	5GiB
64	20GiB	10GiB

序列长度为 1024 时 Llama-2-70B 的 KV 缓存大小

目录

CONTENTS

单GPU推理评估

在某些情况下，批处理可能没有意义。例如，如果你正在构建一个面向用户的聊天机器人，你的产品对延迟更加敏感，因此不能等待批处理填满后再运行推理。在这种情况下，我们应该怎么办？

一种选择是意识到我们将无法充分利用GPU的芯片显存，并进行缩小。例如，我们可以切换到T4 GPU，它具有16 GB的VRAM。这仍然可以容纳我们的7B参数模型，但用于批处理和KV缓存的剩余容量要少得多，只有2 GB。

然而，T4 GPU通常比A10慢。而A100虽然更强大，但也更昂贵。我们可以通过计算一些简单的推理时间下限来量化这种差异。

单token生成时间计算

在生成的自回归部分，如果批量大小为1，我们将受到显存带宽限制。让我们快速计算一下使用以下方程生成单个token所需的时间：

time/token = 总字节数 (模型参数) / 显存带宽

- T4: (2 * 7B) bytes / (300 GB/s) = 46 ms/token
- A10: (2 * 7B) bytes / (600 GB/s) = 23 ms/token
- A100 SXM 80 GB: (2 * 7B) bytes / (2039 GB/s) = 6 ms/token

这些数字仅为近似值，因为它们假设在推理过程中GPU内部没有任何通信，每次前向传递都没有额外开销，并且计算过程完全并行化。

提示词时间计算

在每个GPU上使用批处理提示令牌进行预填充。我们还可以计算填充部分所需的时间，假设我们将所有提示token批处理到单个前向传递中。为了简单起见，假设提示有350个token，并且限制瓶颈是计算而不是显存。

Prefill time = tokens 数量 * (参数量 / 计算带宽)

- T4: 350 * (2 * 7B) FLOP / 65 TFLOP/s = 75 ms
- A10: 350 * (2 * 7B) FLOP / 125 TFLOP/s = 39 ms
- A100 SXM 80 GB: 350 * (2 * 7B) FLOP / 312 TFLOP/s = 16 ms

总生成时间计算

假设生成 150个 token（并且抑制任何停止token），我们的总生成时间如下。

总时间 = prefill time + tokens 数量 * time/token

- T4 = 75 ms + 150 tokens * 46 ms/token = 6.98 s
- A10 = 39 ms + 150 tokens * 23 ms/token = 3.49 s
- A100 SXM 80 GB: 16 ms + 150 tokens * 6 ms/token = 0.92s

A100 和 H800 算力

A100：

- 如果不进行矩阵乘法，则只能实现 19.5 TFLOPS
- 矩阵乘法 FP16 可以实现 312 TFLOPS
- 如果按照稀疏矩阵算是 624 TFLOPS

规格	A100 SXM	A800 SXM	H100 SXM	H800 SXM
双精FP64	9.7 TFLOPS		34 TFLOPS	1TFLOPS
双精FP64 Tensor Flow	19.5 TFLOPS		67 TFLOPS	1TFLOPS
单精FP32	19.5 TFLOPS		67 TFLOPS	
单精TF32	156 TFLOPS | 312 TFLOPS*		495 TFLOPS | 989 TFLOPS*	
半精FP16	312 TFLOPS | 624 TFLOPS*		990 TFLOPS | 1,979 TFLOPS*	
FP8	NA		3,958 TFLOPS	
显存大小	80G		80G	
显存带宽	2039 GB/s		3.35TB/s	
互联带宽	600 GB/s	400GB/s	900GB/s	400GB/s
NVLINK链路	12条	8条	18条	8条
功耗	400W	400W	700W（最高）	

ChatGLM2/3 架构

- d，可以表示为d_head，是单个注意力头的维度
  - 对于 6B d =
- n_heads是注意力头的数量
  - 对于 6B n_heads =
- n_layers是注意力块出现的次数
  - 对于 6B n_layers =
- d_model是模型的维度。d_model = d_head * n_heads
  - 对于 6B d_model =

ChatGLM3 6B 所需 A100 推理卡数量测算

需根据实测数据修订

指标	数值	备注
A: 模型参数量(亿个)	60	展示以 chatGLM 6B 为例
D: 平均实时并发（个）	500	同时使用量
E: 平均每秒 tokens 输出（个）	20	假设平均每秒 20 个左右。
F: 峰值预留倍数	10	综合考虑并发峰值，以及显存预留等因素
G: "算力利用率"	15%	以显存带宽为代表的因素拖累算力利用率
H: 所需算力储备（FLOPS）	8E+15	2*A*(D*E*F)/G
假设精度	所需 A100(块)	A100 算力
TF32	25.64	312 (TFLOPS=10^12FLOPS)　H/312 TFLOPS
TF16	12.82	624 (TFLOPS=10^12FLOPS)　H/624 TFLOPS
INT8	6.41	1248 (TFLOPS=10^12FLOPS)　H/1248 TFLOPS

ChatGLM3 6B 所需 H800 推理卡数量测算

需根据实测数据修订

指标	数值	备注
A: 模型参数量(亿个)	60	展示以 ChatGLM 6B 为例
D: 平均实时并发（个）	500	同时使用量
E: 平均每秒 tokens 输出（个）	20	假设平均每秒 20 个左右。
F: 峰值预留倍数	10	综合考虑并发峰值，以及显存预留等因素
G: "算力利用率"	15%	以显存带宽为代表的因素拖累算力利用率
H: 所需算力储备（FLOPS）	8E+15	2*A*(D*E*F)/G
假设精度	所需 H800(块)	H 800 算力
TF32	8.09	989 (TFLOPS=10^12FLOPS)　H/989 TFLOPS
TF16	16.18	1979 (TFLOPS=10^12FLOPS)　H/1979 TFLOPS
INT8	32.36	3958 (TFLOPS=10^12FLOPS)　H/3958 TFLOPS

Thank You