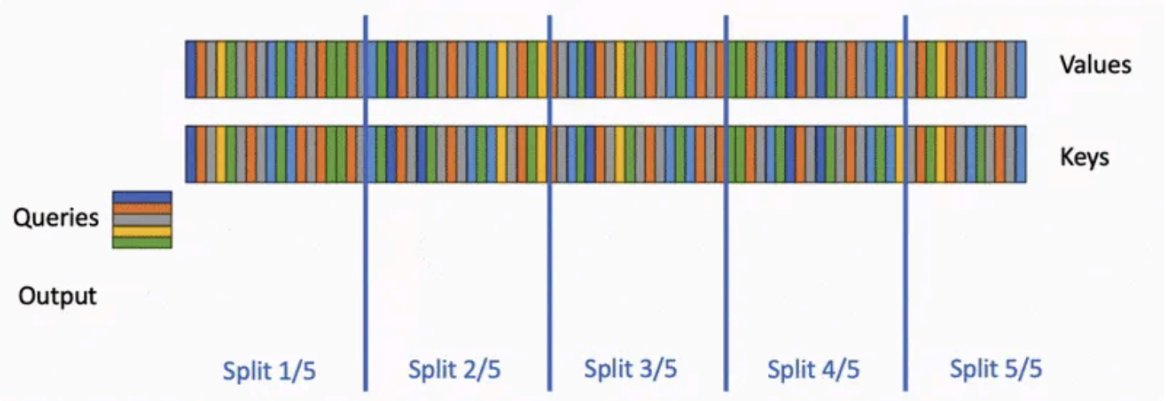
12. FlashDecoding在FlashAttention2上做了哪些改进?

Flash-Decoding借鉴了FlashAttention的优点，**Flash-Decoding在此基础上增加了一个新的并行化维度：keys/values的序列长度。**即使batch size很小，但只要上下文足够长，它就可以充分利用GPU。与FlashAttention类似，Flash-Decoding几乎不用额外存储大量数据到全局内存中，从而减少了内存开销。



Flash Decoding主要包含以下三个步骤：

- 将keys和values分成较小的block
- **使用FlashAttention并行计算query与每个block的注意力（这是和FlashAttention最大的区别）**。对于每个block的并行（因为一行是一个特征维度），Flash Decoding会额外记录attention values的log-sum-exp（标量值，用于第三步进行rescale）
- 对所有output blocks进行reduction得到最终的output，需要用log-sum-exp值来重新调整每个块的贡献

实际应用中，第一步的数据分块不涉及GPU操作（因为不需要在物理上分开），只需对第2步和第3步执行单独的kernels。虽然最终的reduction操作会引入一些额外的计算，但在总体上，Flash-Decoding通过增加并行化的方式取得了更高的效率。

13. FlashDecoding++做了什么优化?

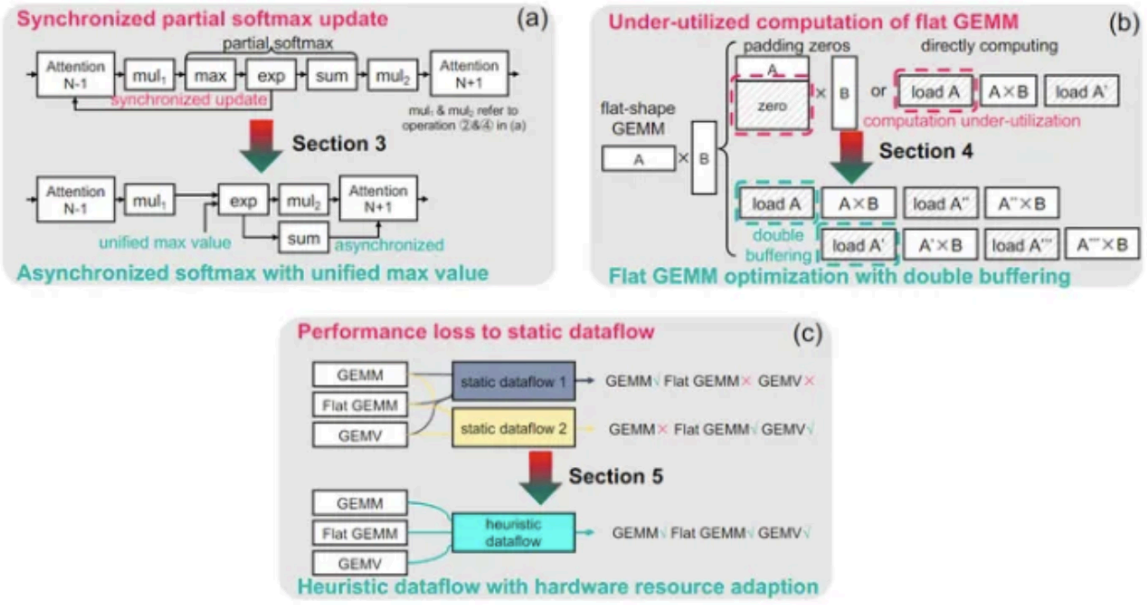
为了提高softmax并行性，之前方法（FlashAttention、FlashDecoding）将计算过程拆分，各自计算partial softmax结果，最后需要通过同步操作来更新partial softmax结果。例如FlashAttention每次计算partial softmax结果都会更新之前的结果，而FlashDecoding是在最后统一更新所有partial softmax结果。

论文分析在A100 GPU上分析了输入长度为1024的情况，这种同步partial softmax更新操作占Llama2-7B推理的注意力计算的18.8%。

这是LLM推理加速的第一个挑战。此外，本文还提出了两个挑战：

- 在解码阶段，Flat GEMM操作的计算资源未得到充分利用。这是由于解码阶段是按顺序生成token，GEMM操作趋于flat-shape，甚至batch size等于1时变成了GEMV（General Matrix-Vector Multiplication）。当batchsize较小时，cublas和cutlass会将矩阵填充zeros以执行更大的batchsize的GEMM，导致计算利用率不足50%。
- 动态输入和固定硬件配置影响了LLM推理的性能。例如，当batchsize较小时，LLM推理的解码过程是memory-bounded，而当batchsize较大时是compute-bounded。

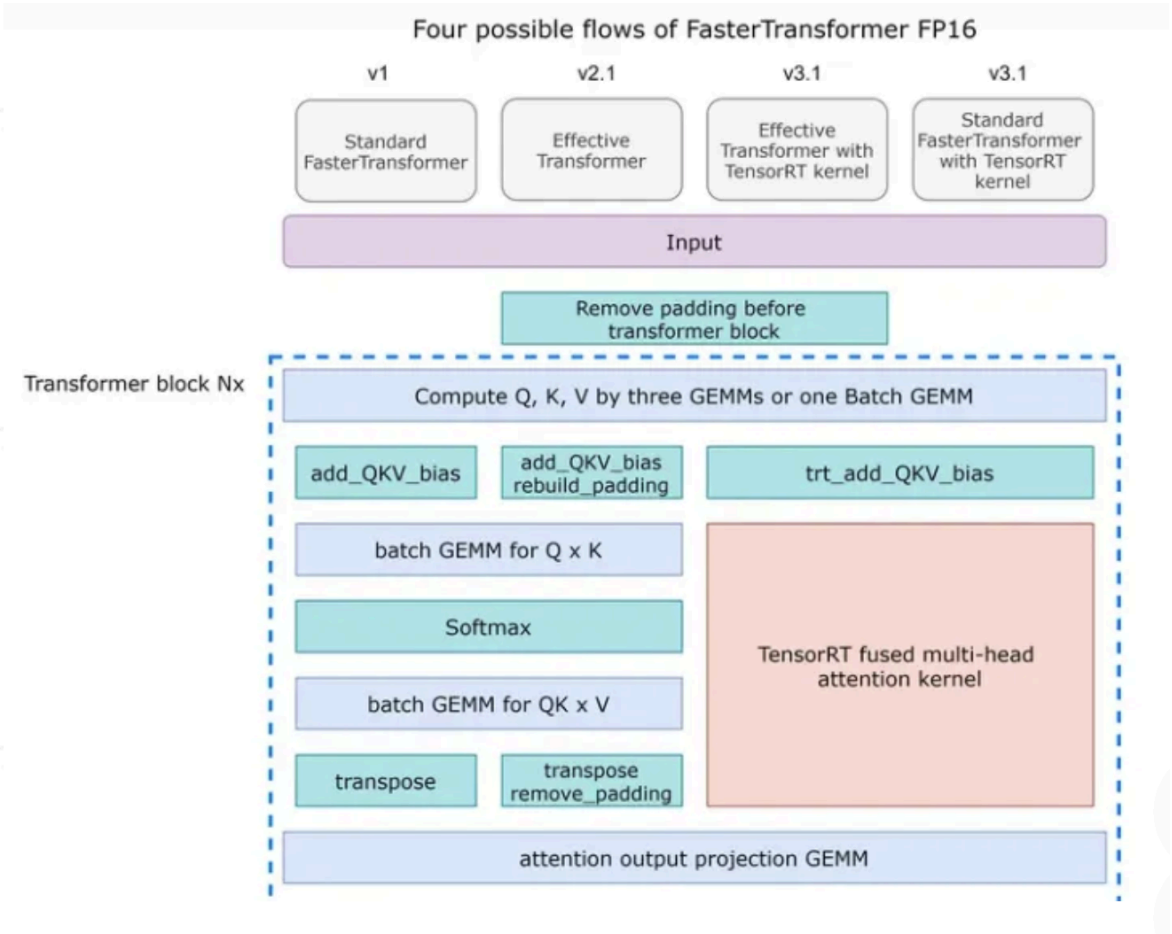
针对这3个问题，flashdecoding++分别提出了对应优化方法：



- FlashDecoding++为分块softmax计算设置了一个共享的最大值。这样可以独立计算partial softmax，无需同步更新。
- FlashDecoding++只将矩阵大小填充到8，对比之前针对flat-shaped GEMM设计的64，提高了计算利用率。论文指出，具有不同shape的flat GEMMs面临的瓶颈也不同，于是进一步利用双缓冲等技术提高kernel性能。
- FlashDecoding++同时考虑了动态输入和硬件配置，针对LLM推理时数据流进行动态kernel优化。

14. 什么是子图融合优化技术，为什么他可以提升推理速度?

图融合技术即通过将多个OP（算子）合并成一个OP，来减少Kernel的调用。因为每一个基本OP都会对应一次GPU kernel的调用，和多次显存读写，这些都会增加大量额外的开销。例如FastTransformer推理框架，图融合是它的一个重要特性，将多层神经网络组合成一个单一的神经网络，将使用一个单一的内核进行计算。这种技术减少了数据传输并增加了数学密度，从而加速了推理阶段的计算。例如，multi-head attention块中的所有操作都可以合并到一个内核中。



15. 大模型稀疏化压缩有哪些技术?

实现大模型稀疏（LLM Sparsity）的一个重要方法是剪枝（Pruning）。剪枝是在保留模型容量的情况下，通过修剪不必要的模型权重或连接来减小模型大小。它可能需要也可能不需要重新培训。修剪可以是非结构化或结构化的。

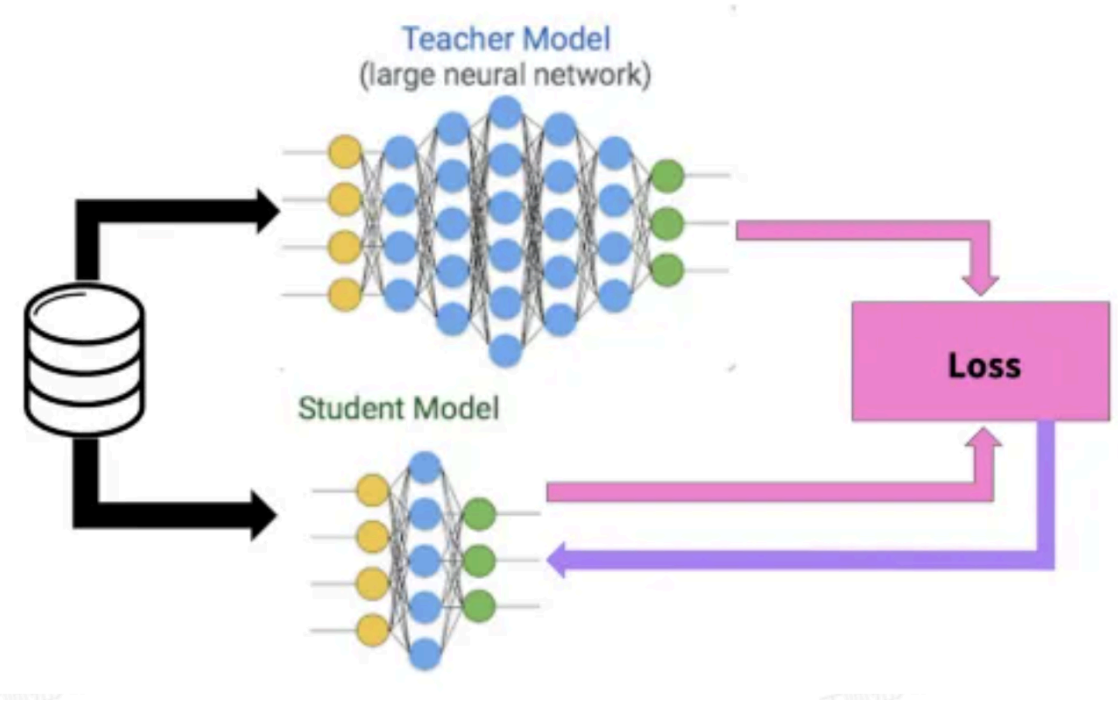
- 非结构化剪枝允许删除任何权重或连接，因此它不保留原始网络架构。非结构化剪枝通常不适用于现代硬件，而且不会带来实际的推理加速。
- 结构化剪枝旨在维持某些元素为0的密集矩阵乘法形式。它们可能需要遵循某些模式限制才能使用硬件内核支持的内容。当前的主流方法关注结构化剪枝，以实现Transformer模型的高稀疏性。

除此之外，还有其他的稀疏化方法，例如：

- SparseGPT：该方法的工作原理是将剪枝问题简化为大规模的稀疏回归实例。它基于新的近似稀疏回归求解器，用于解决分层压缩问题，其效率足以在几个小时内使用单个GPU在最大的GPT模型上执行。同时，SparseGPT准确率足够高，不需要任何微调，剪枝后所损耗的准确率也可以忽略不计。
- LLM-Pruner：遵循经典的"重要性估计-剪枝-微调"的策略，能够在有限资源下完成大预言模型的压缩，结果表明即使剪枝20%的参数，压缩后的模型保留了93.6%的性能。
- Wanda：该方法由两个简单但必不可少的组件构成——剪枝度量和剪枝粒度。剪枝度量用来评估权重的重要性，然后按照剪枝粒度进行裁剪。该方法在65B的模型上只需要5.6秒就可以完成剪枝，同时达到SparseGPT相近的效果。

16. 介绍一下大模型蒸馏技术的原理

知识蒸馏是一种构建更小、更便宜的模型（student模型）的直接方法，通过从预先训练的昂贵模型中转移技能来加速推理（teacher模型）融入student。除了与teacher匹配的输出空间以构建适当的学习目标之外，对于如何构建student架构没有太多限制。



给定数据集，训练student模型通过蒸馏损失来模仿teacher的输出。通常神经网络有一个softmax层，例如LLM输出token的概率分布。将softmax之前的logits层表示为zt和zs，分别表示teacher和student模型。蒸馏损失最小化两个softmax输出之间的差异（温度T）。当标签y已知，可以将其与student的logits之间计算交叉熵，最后将两个损失相加，如下：

$$L_{KD} = L_{distill}(softmax(z_t, T), softmax(z_s, T)) + \lambda L_{CE}(y, z_s)$$

在大模型时代，蒸馏可以与量化、剪枝或稀疏化技术相结合，其中teacher模型是原始的全精度密集模型，而student模型则经过量化、剪枝或修剪以具有更高的稀疏级别，以实现模型的小型化。

17. 无限长的输入对LLM意味着什么，StreamingLLM是如何解决的?

使用LLM处理无限长文本是一项挑战。存储之前所有的键和值（KV）状态需要大量内存，模型可能难以生成超过其训练序列长度的文本。StreamingLLM可以解决这个问题，它只保留最新的标记和注意力汇，而丢弃中间的标记。

18. 在StreamingLLM中，LLM的上下文窗口是否扩大了?

不会。上下文窗口保持不变，只保留最近的标记和注意力汇，丢弃中间的标记。这意味着模型只能处理最新的标记。上下文窗口仍然受到初始预训练的限制。例如，如果Llama-2的预训练上下文窗口为4096个token，那么Llama-2上StreamingLLM的最大缓存大小仍为4096。

19. 我可以将大量文本（如一本书）输入到StreamingLLM中进行摘要吗?

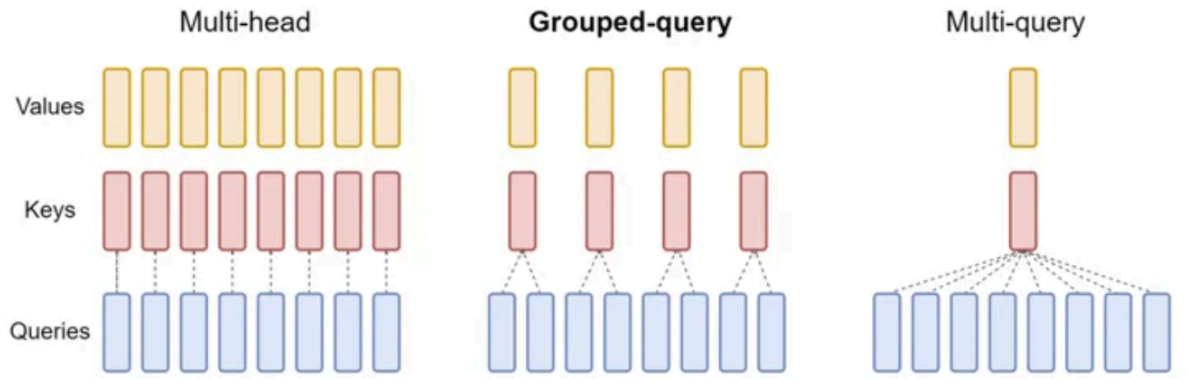
虽然可以输入长篇文本，但模型只能识别最新的标记。因此，如果输入的是一本书，StreamingLLM可能只会对结尾段落进行摘要，而这些段落的内容可能并不深刻。正如前面所强调的，我们既没有扩大LLM的上下文窗口，也没有增强它们的长期记忆。StreamingLLM的优势在于无需刷新缓存就能从最近的标记生成流畅的文本。

20. StreamingLLM的理想用例是什么?

StreamingLLM针对多轮对话等流式应用进行了优化。**它非常适合模型需要持续运行而不需要大量内存或依赖过去数据的场景。**基于LLM的日常助手就是一个例子。StreamingLLM可以让模型持续运行，根据最近的对话做出响应，而无需刷新缓存。早期的方法要么需要在对话长度超过训练长度时重置缓存（丢失最近的上下文），要么需要根据最近的文本历史重新计算KV状态，而这可能会非常耗时。

21. MHA、GQA、MQA推理优化技术的区别是什么?

MQA，全称Multi Query Attention，而GQA则是前段时间Google提出的MQA变种，全称Group-Query Attention。MHA（Multi-head Attention）是标准的多头注意力机制，h个Query、Key和Value矩阵。MQA让所有的头之间共享同一份Key和Value矩阵，每个头只单独保留了一份Query参数，从而大大减少了Key和Value矩阵的参数量。GQA将查询头分成N组，每个组共享一个Key和Value矩阵。



如上图，GQA和MQA都可以实现一定程度的Key value的共享，从而可以使模型体积减小，GQA是MQA和MHA的折中方案。这两种技术的加速原理是（1）减少了数据的读取（2）减少了推理过程中的KV Cache。需要注意的是GQA和MQA需要在模型训练的时候开启，按照相应的模式生成模型。

22. Paged Attention是如何有效管理具有分页的KV缓存的?

KV缓存被静态地"过度配置"，以适应可能的最大输入大小（支持的序列长度），因为输入大小是不可预测的。例如，如果模型支持的最大序列长度为2048，则无论请求的输入大小和生成的输入大小如何，都会在内存中保留一个大小为2048的保留空间。此空间可能是连续分配的，并且通常很大部分未使用，导致了内存浪费或碎片化。此保留空间与请求的存活时间绑定。

如图所示，显示了由于过度配置和KV缓存的低效管理导致的内存浪费和碎片化的示例。1）"保留"表示为将来使用的内存，该内存在请求持续时间内保留。2）"内部碎片化"发生，因为很难预测生成的长度，因此会过度配置内存以适应最大序列长度。3）"外部碎片化"表示由于批次中的请求需要不同的预分配大小而导致的低效。

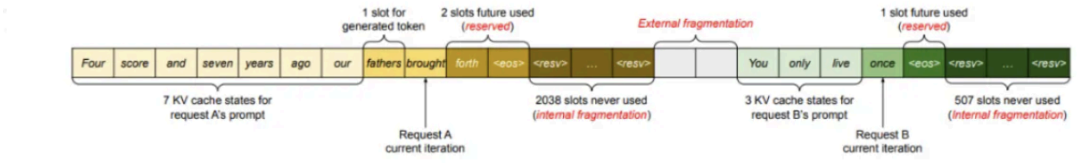


Figure 8. An illustration of memory wastage and fragmentation due to over-provisioning and inefficient KV cache management. Credit: Efficient Memory Management for Large Language Model Serving with PagedAttention

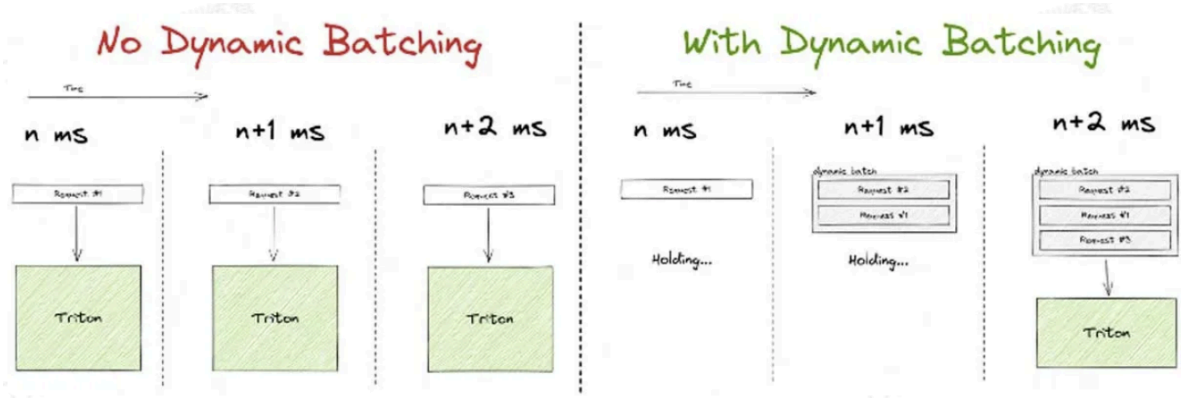
受操作系统中的分页的启发，PagedAttention算法使得可以将连续的键和值存储在内存中的非连续空间中。它将每个请求的KV缓存分为表示固定数量的令牌的块，这些令牌可以非连续存储。

在注意力计算过程中，使用一个块表按需获取这些块。当生成新的令牌时，将进行新的块分配。这些块的大小是固定的，消除了由于不同请求需要不同分配而引起的低效性。这大大减少了内存浪费，增加了批量大小（从而提高吞吐量）。

23. 介绍一下动态批处理技术

LLM具有一些独特的执行特性，可能使得在实践中难以有效地批处理请求。单个模型可以同时用于多种看起来非常不同的任务。从聊天机器人地简单问答响应到文档的摘要或长代码块的生成，工作负载是高度动态的，输出的大小相差几个数量级。

这些多样性可能使得分批处理请求并有效地并行执行成为一种具有挑战性的优化方法，但这可能导致一些请求比其他请求提前完成。



为了管理这些动态负载，许多LLM服务器解决方案包括一种优化的调度技术，称为连续或在飞行批处理。它利用了将LLM的整体文本生成过程分解为对模型的多次执行迭代的事实。

使用在飞行中批处理时，服务器运行时在整个批处理完成之前就开始将完成的序列从批次中删除。然后它开始执行新的请求，而其他请求仍在进行中。在实际使用案例中，在飞行中批处理可以大大增加GPU的整体利用率。