## High-throughput Generative Inference of Large Language Models with a Single GPU

### 1 Introduction

#### 1.1 Motivation：

近些年大规模语言模型在各种任务中表现出高性能的优势，模型的参数可以有数十亿甚至是数万亿，这导致模型运行时对计算和内存的要求极高。例如：GPT-175B需要325GB的GPU显存来load模型的weight；需要5个A100（80G）和复杂的并行策略。因此，降低大规模语言模型的推理资源需求引起了广泛的兴趣。

本文聚焦于面向吞吐量的setting（比如：benchmarking、information extraction、data wrangling、form processing）。这些任务的关键特征是：它们在大量的token上以batch来运行推断，而对token生成过程的延迟不敏感。这个特点为用延迟来换取更高的吞吐量、减少资源需求提供了可能。

#### 1.2 Related work

LLM的发展，使得LLM推断成为了重要的工作，无论系统还是算法都有很多研究。

在以前，降低大规模语言模型（LLM）的工作对应于三个方向：（1）模型压缩以降低总内存占用；（2）通过去中心化，协作推理来分摊推理成本；（3）offloading to 利用CPUI和disk的内存。这些技术极大地降低了使用LLM的资源需求，但是也有着明显的局限性。在（1）和（2）两个技术中，研究人员通常假设模型适合GPU内存，因此很难用单GPU运行175B规模的模型；另一方面，由于低效的I/O调度和tensor placement，基于offload的系统在单GPU上很难实现可以接受的吞吐量。

本文的工作：这篇文章聚焦于offload策略，单GPU、高吞吐量。对于有限的GPU内存，将模型offload到二级存储上（CPU），并通过部分的load部分的执行计算。在面向吞吐量的场景中，可以通过使用large batch size来牺牲延迟，并将昂贵的I/O操作分摊到与计算overlap的large batch输入的不同内存之间。

#### 1.3 Challenge

但是作者说，尽管牺牲了延迟，在有限的GPU资源下实现高吞吐量的生成推理也有有挑战的。

第一，设计高效的offload策略。模型中有三种tensor：weights、activations和key-value cache。该策略应该指定what tensor to offload; where to offload(CPU/disk); when to offload。模型中The batch-by-batch,token-by-token, and layer-by-layer的结构构成了复杂的依赖图，有着多种计算方式，构成了复杂的设计空间。现在的基于offload的系统是从训练中找寻offload策略，这种策略被证明了只是次优点，执行了大量I/O操作，实际的吞吐量也很低。

第二，开发高效的压缩策略。先前的工作压缩了LLM的weights和activations，有了一定的成果。但是，当压缩与offload相结合以实现高吞吐量的时候，weights和KV cache的I/O成本和内存浪费变得非常严重，这motivate了替代压缩的方案。

#### 1.4 Contribution

提出了FlexGen，一个实现高吞吐量LLm推理的offload框架。聚集了来此GPU、CPU、disk的memory，并有效调度I/O操作，以及可能的压缩方法和分布式的流水线并行性。

第一，通过考虑计算调度、张量placement和计算delegation，形式化的定义了offloading策略的搜索空间。证明了该搜索空间可以捕获I/O复杂度在2个最优值范围内的计算顺序。然后，开发了一个基于线性规划的搜索算法来优化搜索空间内的吞吐量。该算法可以针对各种硬件规格进行配置，并可以轻松扩展，以结合延迟和吞吐量约束，从而帮助平稳地在权衡空间中寻找最优策略。与现有的策略相比，该方案统一了weights、activations和KV cache的placement，从而实现显著更高的批处理上限，这是实现高吞吐量的关键。

第二，可以将OPT-175B等LLMs的weights和KV cache压缩到4bits，而无需重新训练或校准，所有这些都具有可忽略不计的精度损失。这是通过细粒度的分组量化实现的(Shen等人，2020)，这适用于减少offload期间的I/O成本和内存使用。

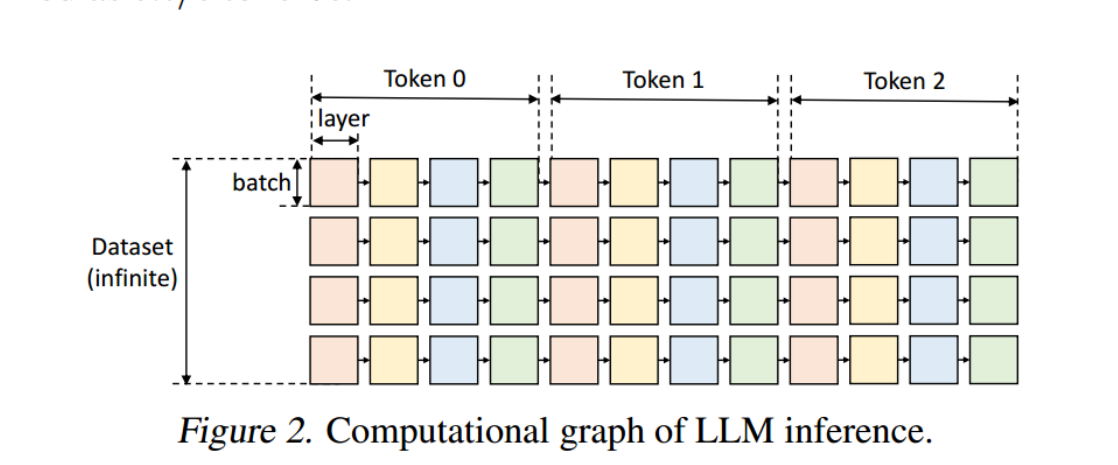
最后，在NVIDIA T4（16G）上运行OPT-175B来演示了FlexGen的效率。与DeepSpeed Zero-Inference和hugging Face Accelerate这两个最先进的基于卸载的推理系统相比，FlexGen通常允许批处理规模大几个数量级。因此，FlexGen可以实现更高的吞吐量。在单个T4 GPU、208G CPU和1.5T SSD，输入序列长度为512，输出序列长度为32：在同样的5000s延迟下，FlexGen（有效batch size大小为64，or2048个token）可以实现比DeepSpeed Zore-Inference（batch size为1，or32个token）高出40X的吞吐量，而Hugging Face Accelerate无法完成一个batch；当延迟为12000s时，FlexGen实现了比baselines高69X的吞吐量，因为它可以将有效batch size扩大到256（8192个token），而DeepSpeed Zero-Inference和hugs Face Accelerate由于内存不足问题无法使用大于2的批大小；如果允许4bits的压缩，FlexGen可以达到100X多的最大吞吐量，有效batch size 144（4608个token），延迟4000s，没有使用ssd offload。

### 2 卸载策略

#### 2.1 问题定义

在GPU、CPU、disk设备中，disk不能进行计算。当LLM无法完全装入GPU时，需要将其卸载到二级存储中，并part-by-part的加载LLM来诸葛执行计算。

这可卸载过程相当于一个图遍历问题。如选图所示：相同的layer层是共享参数的；



•一个正方形只有在同一行中它左边的所有正方形都被计算出来时才能被计算出来。

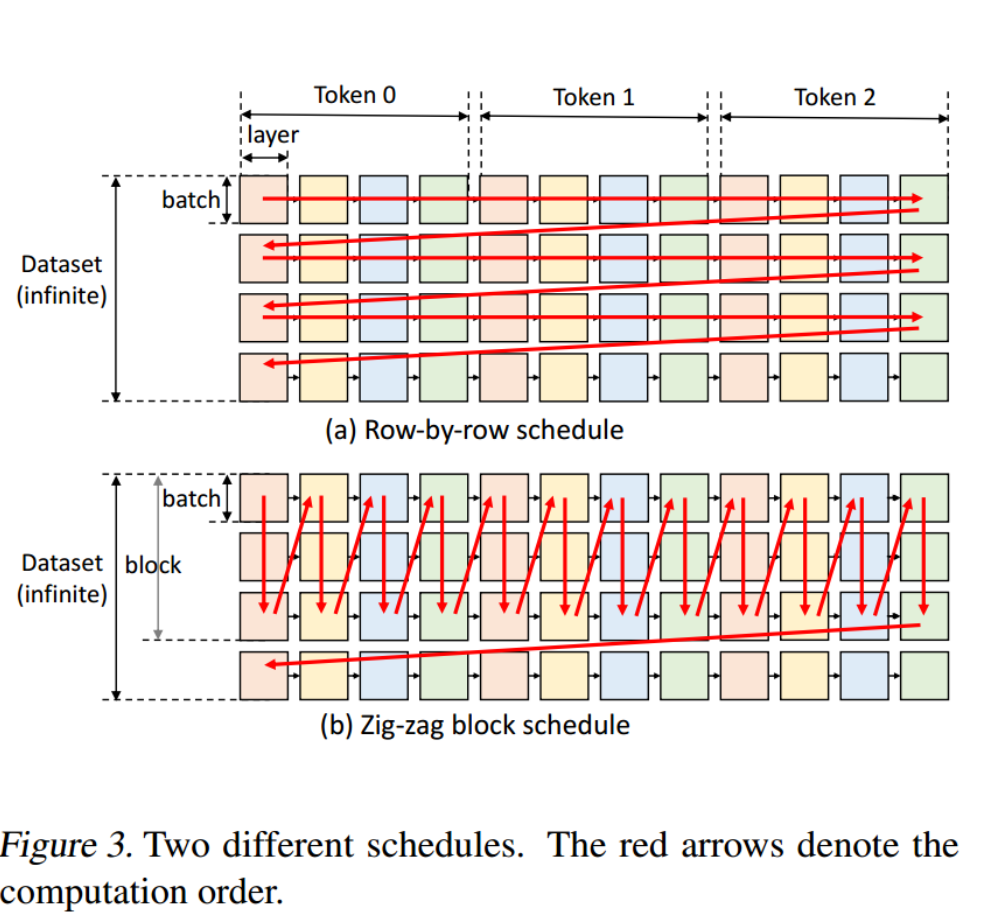
•要在一个设备上计算一个正方形，它的所有输入(权重、激活、缓存)必须加载到同一个设备上。

•经过计算后，一个方块产生两个输出:激活和KV缓存。应该存储激活，直到计算它的右兄弟。KV缓存应该一直存储到计算出同一行上最右边的方块为止。

•在任何时候，存储在设备上的张量的总大小都不能超过其内存容量。

目标就是找到一个有效的路径，使总执行时间最小化，其中包括在设备之间移动张量时的计算成本和I/O成本。

#### 2.2 搜索空间

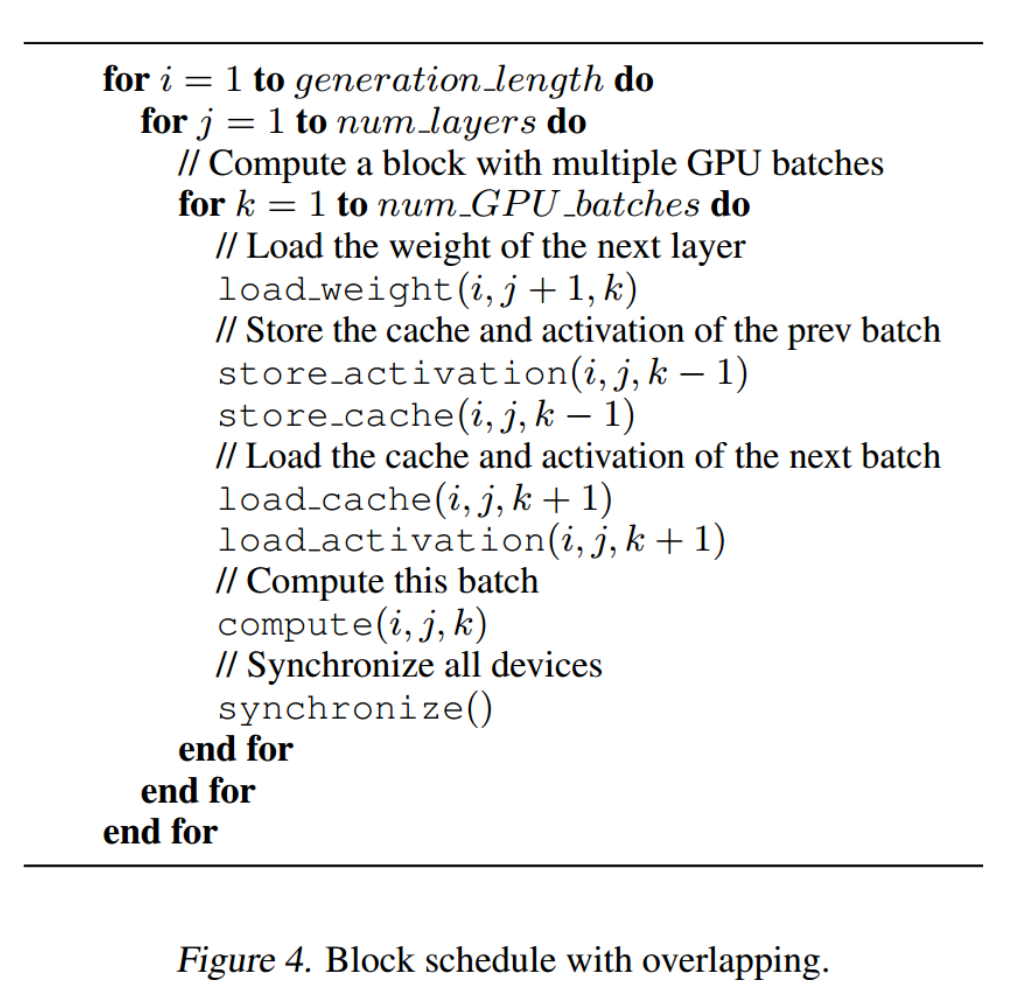


计算schedule：显然，可以逐行/逐列的遍历。现在的所有的系统都是逐行遍历，这是因为这是完成一个batch的最快方式，并且KV cache可以在一行时候立即释放。然而，由于不同的layer之间不共享权重，这个调度产生了巨大的I/O成本。

块调优：为了减少这个I/O成本，可以逐列遍历图。一列中的所有方块共享weight，然而不能一下在一列中遍历到最后，因为激活和KV cache需要训处，当它填满CPU和磁盘内存时，必须停止。如3-b所示。除了这个，还有一种更高级别的I/O最有调度，论文没有实现。在附录2中证明了这种简单的块调度只比最优调度慢两倍（文中的定理4.1）。

Overlap：另一个典型的优化时overlap。Overlap下一层weiget、下一batch的cache/activations、上一batch的cache/activations和当前batch的计算。将这种重叠添加到块调优中。最内层循环中的前六个函数在六个线程中并行启动，因为他们之间没有依赖。最后一个函数是同步这六个线程。依靠操作系统和CUDA来解析底层硬件资源的调度。

两个参数：GPU的batch大小和block内的batch数量。



张量放置：策略还应该描述怎样存储这些张量。定义了三个参数wg、wc、wd描述GPU、CPU、disk上的weight百分比。类似的：hg、hc、hd：activations；cg、cc、cd：KV cache。以权重张量为例，从粗粒度到细粒度，我们可以在模型粒度上划分权重(例如，将模型中50%的层分配给GPU)，在层粒度上划分权重(例如，将一层中50%的张量分配给GPU)，或者在张量粒度上划分权重(例如，将张量中50%的元素分配给GPU)。粒度越粗，运行开销越小，但是灵活性较差而且成本难以分析。**考虑到运行时开销和所需的灵活性，我们使用层粒度作为权重，张量粒度用于激活和KV缓存**。

计算delegation：虽然CPU比GPU慢很多，但是某些情况下使用CPU计算仍然是有益的。在decoder过程中，注意力系数收到I/O的限制，而KV cache存储在CPU上，在GPU上计算注意力系数需要将整个KV cache移动到GPU上，导致了大量的I/O成本，而在CPU上计算注意力系数不需要移动KV cache，只需将activations从GPU移动到CPU。定量：（设b为GPU批大小，s为序列长度，h1为隐藏大小）移动的KV cache大小为b × s× h1 × 4字节，移动的激活大小为b × h1 × 4字节，因此在CPU上计算注意力系数减少了s× I/O。对于较长的序列s>512，如果关联的KV cache存储在CPU或disk上，则最好在CPU上计算注意力系数。

#### 2.3 cost model和搜索策略

Cost model预测在一个块中，一层预填充的延迟为Tpre，一层解码时的平均延迟为Tgen。计算总延迟可以估计为：T=Tpre\*l+Tgen\*（n-1）\*l。l（layer num）、n（token num）。

假设overlap完全，Tpre可估计为Tpre = max(ctogp, gtocp, dtocp, ctodp, compp)，其中ctogp, gtocp, dtocp, ctodp, compp分别表示一层预填充时CPU读到GPU、GPU写到CPU、磁盘读到CPU、CPU写到磁盘、计算的时延。

同样，Tgen可以估计为Tgen = max(ctogg, gtocg, dtocg, ctodg, compg)，其中ctogg, gtocg, dtocg, ctodg, compg分别表示一层解码时从CPU读到GPU、从GPU写到CPU、从磁盘读到CPU、从CPU写到磁盘、计算的时延。

一个策略包含11个变量:块大小bls, GPU批量大小gbs，权重位置wg, wc, wd，激活位置hg, hc, hd, KV缓存位置cg, cc, cd。在实际操作中，百分比不能是0到1之间的任意实数，因为张量不能被任意分割。但是，我们将成本模型中的百分比变量放宽为0到1之间的任何实数，因为它是逐渐变化的。我们把这个问题作为一个两级优化问题来解决。我们首先枚举(bls, gbs)元组的几个选项。一般来说，gbs是4的倍数，bls小于20，所以没有太多的选择。然后在固定的bls, gbs下，找到最佳位置p = (wg, wc, wd, cg, cc, cd, hg, hc, hd)就变成了一个线性规划问题，如Eq.(1)所示。线性规划问题可以很快解决，因为只有9个变量。这个公式也可以灵活地扩展，包括延迟约束和模型近似方法，如压缩。

#### 2.4 扩展到多GPU

有两种模型并行性:张量并行性和流水线并行性(Narayanan et al, 2021;Zheng等，2022)。张量并行可以减少单查询延迟，而流水线并行由于通信成本低，可以在吞吐量上实现良好的扩展。由于我们的目标是吞吐量，FlexGen实现了流水线并行。

通过在m个gpu上平等地划分一个l层LLM来使用流水线并行性，然后所有gpu的执行都遵循相同的模式。问题简化为在一个GPU上运行n/m层转换器。我们可以直接重用为一个GPU开发的策略搜索。为了实现微批处理管道，在图4中添加了一个新的for循环，以组合迭代级管道并行执行计划(Huang et al, 2019;Yu等人，2022)与单设备卸载运行。

### 3 近似方法

#### 3.1 组量化

其它工作的量化主要是为了加速计算，FlenGen的量化是为了减少压缩和节约IO，所以它们计算前会反量化回FP16来计算。所谓的组量化就是把一个tensor在确定维度上的连续g个元素归为一组，然后对这一组做量化。

作者发现压缩到4比特，组大小选64，权重沿着输出维度分组、KV缓存沿着隐藏维度分组，这样能比较好的保留精度。

#### 3.2 系数Attention

作者用了Top-K稀疏近似，先从K cache里找到Top-K个token的下标，按照再按照这个下标去加载V cache并计算。