大模型训练为什么用A100不用4090

原创 李博杰 包包算法笔记 2023-09-12 22:23 发表于北京

作者:李博杰

链接:https://zhuanlan.zhihu.com/p/655402388

包包算法笔记
包大人的算法，程序，机器学习，职场，理财闲谈。
110篇原创内容

公众号

这是一个好问题。先说结论，**大模型的训练用 4090 是不行的，但推理（inference/serving）用 4090 不仅可行，在性价比上还能跟 H100 打个平手。**

事实上，**H100/A100 和 4090 最大的区别就在通信和内存上，算力差距不大。**

	H100	A100	4090
Tensor FP16 算力	1979 Tflops	312 Tflops	330 Tflops
Tensor FP32 算力	989 Tflops	156 Tflops	83 Tflops
内存容量	80 GB	80 GB	24 GB
内存带宽	3.35 TB/s	2 TB/s	1 TB/s
通信带宽	900 GB/s	900 GB/s	64 GB/s
通信时延	~1 us	~1 us	~10 us
售价	$30000~$40000	$15000	$1600

H100 这个售价其实是有 10 倍以上油水的。2016 年我在 MSRA 的时候，见证了微软给每块服务器部署了 FPGA，把 FPGA 打到了沙子的价格，甚至成为了供应商 Altera 被 Intel 收购的重要推手。2017 年我还自己挖过矿，知道什么显卡最划算。后来在华为，我也是鲲鹏、昇腾生态软件研发的核心参与者。因此，一个芯片成本多少，我心里大概是有数的。

鲲鹏的首席架构师夏 Core 有一篇知名文章《谈一下英伟达帝国的破腚》，很好的分析了 H100 的成本：

> 把他的成本打开，SXM 的成本不会高于 300$，封装的 Substrate 及 CoWoS 大约也需要 $300，中间的 Logic Die 最大颗，看上去最高贵：） 那是 4nm 的一颗 814mm2 的 Die，TSMC 一张 12 英寸 Wafer 大致上可以制造大约 60 颗这个尺寸的 Die，Nvidia 在 Partial Good 上一向做得很好（他几乎不卖 Full Good），所以这 60 颗大致能有 50 颗可用，Nvidia 是大客户，从 TSMC 手上拿到的价格大约是 $15000，所以这个高贵的 Die 大约只需要 $300。哦，只剩下 HBM 了，当前 DRAM 市场疲软得都快要死掉一家的鬼样了，即使是 HBM3 大抵都是亏本在卖，差不多只需要 $15/GB，嗯，80GB 的容量成本是 $1200。TSMC 曾经讲过一个故事。台湾同胞辛辛苦苦攒钱建厂，一张 4nm 那么先进的工艺哦，才能卖到 $15000，但是那某个客户拿去噢，能卖出 $1500000（$30000*50）的货啦，机车，那样很讨厌耶。你懂我意思吗？就如最开始说的，在这个世界的商业规则下，$2000 成本的东西卖 $30000，只有一家，销售量还很大，这是不符合逻辑的，这种金母鸡得有航母才守得住。

据说微软和 OpenAI 包下了 H100 2024 年产能的一半，猜猜他们会不会发挥当年跟 Altera 砍价的传统艺能？会真的花 $40,000 * 500,000 = 200 亿美金去买卡？

咱们再分析下 4090 的成本，5nm 的 609mm2 Die，大约成本是 $250。GDDR6X，24 GB，按照 1 GB $10 算，$240。PCIe Gen4 这种便宜东西就算 $100 吧。封装和风扇这些东西，算

它 $300。总成本最多 $900，这样的东西卖 $1600，算是良心价了，因为研发成本也是钱啊，更何况 NVIDIA 的大部分研发人员可是在世界上程序员平均薪酬最高的硅谷。

可以说，H100 就像是中国一线城市的房子，本身钢筋水泥不值多少钱，房价完全是被供求关系吹起来的。我在 LA 已经住了两周，公司租的房子使用面积是我北京房子的 4 倍，但售价只贵了 30%，还带个小院，相当于单位面积的房价是北京的 1/3。我跟本地的老外聊天，他们都很吃惊，你们的平均收入水平比 LA 低这么多，怎么买得起北京的房子的?

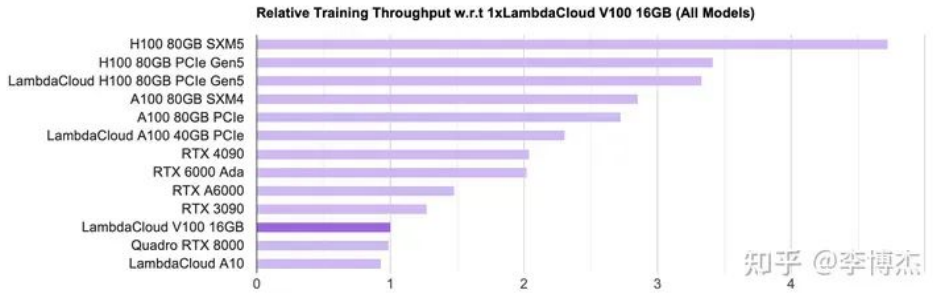
问题来了，如果 4090 这么香的话，为啥大家还要争着买 H100，搞得 H100 都断货了？甚至 H100 都要对华禁售，搞出个 H800 的阉割版?

大模型训练为什么不能用 4090

GPU 训练性能和成本对比

LambdaLabs 有个很好的 GPU 单机训练性能和成本对比，在此摘录如下。

首先看吞吐量，看起来没有什么违和的，在单卡能放下模型的情况下，确实是 H100 的吞吐量最高，达到 4090 的两倍。看算力和内存也能看出来，H100 的 FP16 算力大约是 4090 的 6 倍，内存带宽是 3.35 倍，训练过程中由于 batch size 比较大，大多数算子是 compute bound（计算密集型），少数算子是 memory bound（内存密集型），这个结果是不意外的。



LambdaLabs PyTorch 单卡训练吞吐量对比图

Visualization: table | Metric: throughput | Precision: fp16 | Number of GPUs: 1x | Model: All Models

Search GPUs...

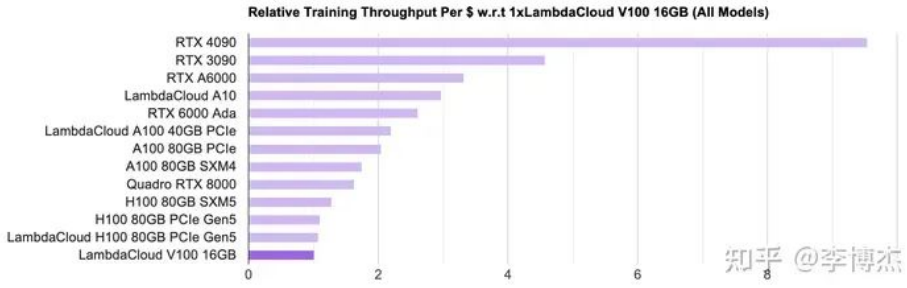
Configuration	ssd	resnet50	gnmt	transformerxlbase	transformerxllarge	tacotron2	waveglow
H100 80GB SXM5	782	2616	467071	62259	30841	78108	393500
H100 80GB PCIe Gen5	566	1866	332521	47919	22586	63991	273768
LambdaCloud H100 80GB PCIe Gen5	549	1851	302013	45433	21874	63200	276658
A100 80GB SXM4	458	1582	243500	43078	17918	59716	244438
A100 80GB PCIe	437	1527	227949	41602	17082	58956	234859
RTX 6000 Ada	341	1146	160184	36745	16049	40331	156904
LambdaCloud A100 40GB PCIe	395	1396	186947	35366	16034	38131	195108
RTX 4090	379	1301	159511	40427	14509	32661	160816
RTX A6000	260	916	116438	26030	11126	25496	116866
Quadro RTX 8000	182	651	88498	18223	7853	18587	43964
RTX 3090	236	905	86080	22863	7821	25018	97100
LambdaCloud V100 16GB	229	833	79111	18493	6071	12611	47253
LambdaCloud A10	174	638	67836	16283	5614	16900	80357

LambdaLabs PyTorch 单卡训练吞吐量对比表

然后看性价比，就有意思了，原来排在榜首的 H100 现在几乎垫底了，而且 4090 和 H100 的差距高达接近 10 倍。这就是因为 H100 比 4090 贵太多了。

由于 H100 货源紧张，云厂商的 H100 租用价格就更黑了，按照标价大约 7 个月就可以回本。就算大客户价能便宜一半，一年半也足够回本了。

在价格战中过惯了苦日子的 IaaS 云服务商看到这样的 H100 回本速度，估计要感叹，这真是比区块链挖矿回本还快呐。



LambdaLabs PyTorch 单卡训练单位成本吞吐量对比图

Visualization: table | Metric: throughput/$ | Precision: fp16 | Number of GPUs: 1x | Model: All Models

Search GPUs...

Configuration	ssd	resnet50	gnmt	transformerxlbase	transformerxllarge	tacotron2	waveglow
RTX 4090	0.178353	0.612235	75.064000	19.024471	6.827765	15.369882	75.678118
RTX 3090	0.085818	0.329091	31.301818	8.313818	2.844000	9.097455	35.309091
RTX A6000	0.058592	0.206423	26.239549	5.865915	2.507268	5.745577	26.336000
RTX 6000 Ada	0.044401	0.149219	20.857292	4.784505	2.089714	5.251432	20.430208
LambdaCloud A10	0.055680	0.204160	21.707520	5.210560	1.796480	5.408000	25.714240
LambdaCloud A100 40GB PCIe	0.037844	0.133749	17.911090	3.388359	1.536192	3.653269	18.692982
Quadro RTX 8000	0.030145	0.107826	14.658054	3.018302	1.300704	3.078592	7.281822
A100 80GB PCIe	0.032888	0.114920	17.155146	3.130913	1.285569	4.436952	17.675183
A100 80GB SXM4	0.028185	0.097354	14.984615	2.650954	1.102646	3.674831	15.042338
H100 80GB SXM5	0.021297	0.071244	12.720231	1.695564	0.839925	2.127197	10.716596
H100 80GB PCIe Gen5	0.018306	0.060353	10.754932	1.549874	0.730513	2.069700	8.854648
LambdaCloud H100 80GB PCIe Gen5	0.017757	0.059868	9.768193	1.469468	0.707484	2.044117	8.948121
LambdaCloud V100 16GB	0.023044	0.083824	7.960855	1.860931	0.610918	1.261631	4.[illegible]

LambdaLabs PyTorch 单卡训练单位成本吞吐量对比表

大模型训练的算力需求

既然 4090 单卡训练的性价比这么高，为啥不能用来做大模型训练呢？抛开不允许游戏显卡用于数据中心这样的许可证约束不谈，从技术上讲，根本原因是大模型训练需要高性能的通信，但 4090 的通信效率太低。

大模型训练需要多少算力？训练总算力（Flops）= 6 * 模型的参数量 * 训练数据的 token 数。

我今年初第一次看到有人煞有介事地讲这个公式的时候，觉得这不是显然的吗？又看到 OpenAI 的高级工程师能拿 90 多万美金的年薪，顿时整个人都不好了，还是 AI 香呀。之前我也面试过一些做 AI 的工程师，包括一些做 AI 系统优化的专家，连 Q、K、V 是啥都说不清楚，LLaMA 每个 tensor 的大小也算不出来，就这样还能拿到 offer。

APNet 2023 panel 的主题是 Network, AI, and Foundational Models: Opportunties and Challenges。前面几个问题都中规中矩的，panelists 有点放不开，我就提了一个问题，网络历史上的重要成就基本上都基于对应用场景深刻的理解，但我们现在做网络的很多都不了解 AI，甚至连每个 tensor 的大小和每个 step 传输的数据量都不知道，如何让 network community 更了解 AI 呢?

这下热闹了，台下的谭博首先发言，说我在华为肯定能知道所有这些东西；然后传雄老师也跟了一句，要是做网络的懂了太多 AI，那可能他就变成一个 AI guy 了。接着主持人陈凯教授问，你们有谁真的训练过大模型？沉默了一会儿，阿里的兄弟先说，我算是半个训练过大模型的，我们做的东西是支撑阿里大模型 infra 的。后面又有 panelist 说，做 AI 系统的网络优化是否有必要自己懂 AI 呢，是不是只要会做 profiling 就行了？

我个人观点仍然是，AI 并不难学，要想做好 AI 系统优化，可以不懂 attention 的 softmax 里面为什么要除以 sqrt(d_k)，但**不能不会计算模型所需的算力、内存带宽、内存容量和通信数据量**。Jeff Dean 就有个很有名的 Numbers Every Programmer Should Know，数量级的估算对任何系统优化来说都很关键，不然根本不知道瓶颈在哪里。

回到大模型训练所需的总算力，其实很简单，**6 * 模型的参数量 * 训练数据的 token 数就是所有训练数据过一遍所需的算力。这里的 6 就是每个 token 在模型正向传播和反向传播的时候所需的乘法、加法计算次数。**

一堆矩阵相乘，简单来想就是左边若干个神经元，右边若干个神经元，组成一个完全二分图。选出其中任意一个左边的神经元 l 和右边的神经元 r。

正向传播的时候：

l 把它的输出乘上 l 和 r 之间的权重 w，发给 r；

r 不可能只连一个神经元吧，总要把多个 l 的加到一起，这就是 reduce，需要一次加法。

反向传播的时候：

r 把它收到的梯度乘上 l 和 r 之间的权重 w，发给 l；

l 也不可能只连一个 r，需要把梯度 reduce 一下，做个加法；

别忘了权重 w 需要更新，那就要计算 w 的梯度，把 r 收到的梯度乘上 l 正向传播的输出（activation）；

一个 batch 一般有多个 sample，权重 w 的更新需要把这些 sample 的梯度加到一起。

一共 3 次乘法，3 次加法，不管 Transformer 多复杂，矩阵计算就是这么简单，其他的向量计算、softmax 之类的都不是占算力的主要因素，估算的时候可以忽略。

想起来我 2019 年刚加入 MindSpore 团队的时候，领导让我开发一个正向算子的反向版本，我求导给求错了，搞得算子的计算结果总是不对，还以为是我们的编译器出 bug 了。当发现求导求错的时候，领导像以为我没学过微积分一样看着我，确实我的微积分学的不好，这也是我从数学专业转到计算机专业的原因之一。

在 MindSpore 的时候，自动微分一共就不到 1000 行代码，按照微分公式递归计算下去就行了，但自动微分作为一个重要特性被吹了半天，我都感觉不好意思了。

模型的参数量和训练数据的 token 数之间也有个比例关系，这也很容易理解，只要把模型想象成数据的压缩版本就行了，压缩比总是有极限的。模型的参数量太小，就吃不下训练数据里面所有的知识；模型的参数量如果大于训练数据的 token 数，那又浪费，还容易导致 over-fitting。

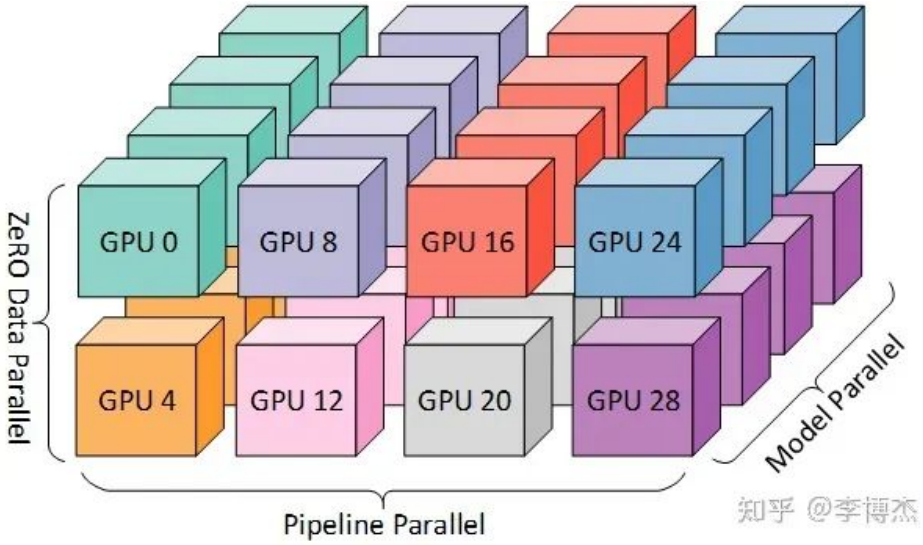
训练 LLaMA-2 70B 需要多少张卡

有了模型训练所需的总算力，除以每个 GPU 的理论算力，再除以 GPU 的有效算力利用比例，就得到了所需的 GPU-hours，这块已经有很多开源数据。LLaMA 2 70B 训练需要 1.7M GPU hours（A100），要是用 1 个 GPU，那得算 200 年。要在一个月这种比较能接受的时间周期内训练出来，就得至少有 2400 块 A100。

如果用 4090，单卡 FP16 算力是跟 A100 差不多（330 vs 312 Tflops），但是内存带宽比 A100 低一半（1 vs 2 TB/s），内存容量更是差好几倍（24 vs 80 GB），计算梯度时需要使用的 TF32 算力也低一半（83 vs 156 Tflops），综合起来 4090 单卡的训练速度还比 A100 稍低（参考前面 LambdaLabs 的评测）。

就按照 2048 块 4090 算吧，这 2048 块 4090 之间的通信就成了最大的问题。

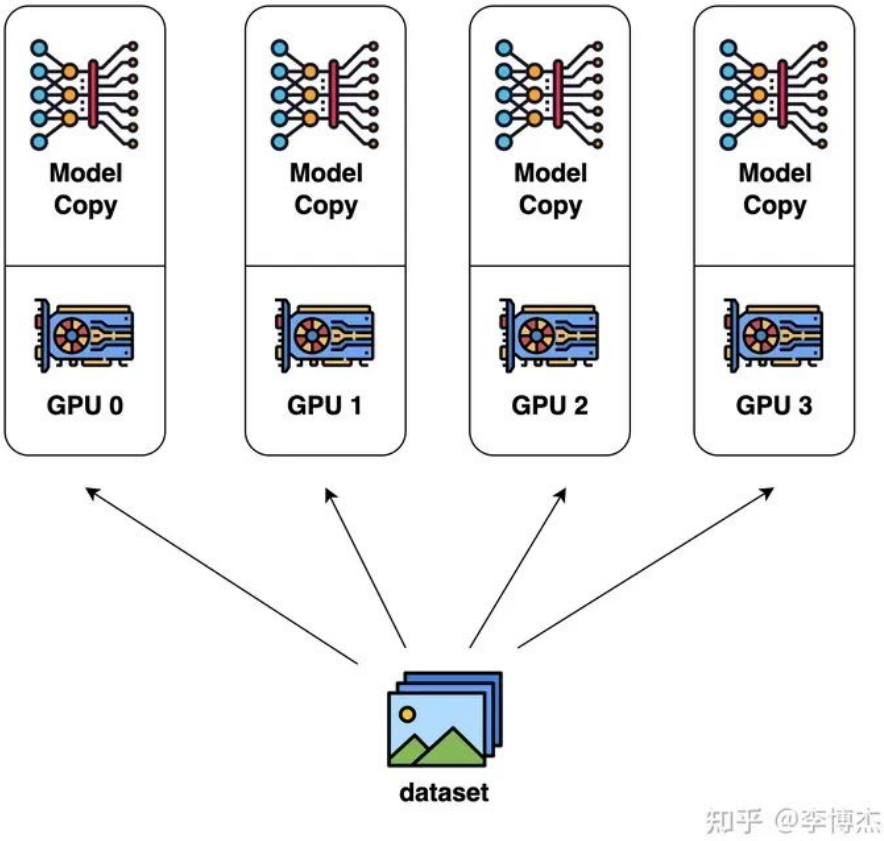
为什么？**一般有 tensor parallelism、pipeline parallelism、data parallelism 几种并行方式，分别在模型的层内、模型的层间、训练数据三个维度上对 GPU 进行划分。**三个并行度乘起来，就是这个训练任务总的 GPU 数量。



三种并行方式从三个维度划分计算空间的示意图，来源：DeepSpeed

Data parallelism（数据并行）

数据并行是最容易想到的并行方式。每个 GPU 分别计算不同的输入数据，计算各自的梯度（也就是模型参数的改变量），再把梯度汇总起来，取个平均值，广播给各个 GPU 分别更新。



Data Parallelism 示意图，来源：Colossal AI

但只用数据并行是肯定不行的，因为一块 GPU 放不下整个 LLaMA 70B 模型。

就模型训练需要多少 GPU 内存，我发现能算清楚的人就不多。有的人甚至以为只需要把模型的参数和反向传播的梯度存下来就够了。事实上，**训练需要的内存包括模型参数、反向传播的梯度、优化器所用的内存、正向传播的中间状态（activation）。**

优化器所用的内存其实也很简单，如果用最经典的 Adam 优化器，它需要用 32 位浮点来计算，否则单纯使用 16 位浮点来计算的误差太大，模型容易不收敛。因此，每个参数需要存 4 字节的 32 位版本（正向传播时用 16 位版本，优化时用 32 位版本，这叫做 mixed-precision），还需要存 4 字节的 momentum 和 4 字节的 variance，一共 12 字节。如果是用类似 SGD 的优化器，可以不存 variance，只需要 8 字节。

正向传播的中间状态（activation）是反向传播时计算梯度必需的，而且跟 batch size 成正比。Batch size 越大，每次读取模型参数内存能做的计算就越多，这样对 GPU 内存带宽的压力就越小。可是不要忘了，正向传播的中间状态数量是跟 batch size 成正比的，GPU 内存容量又会成为瓶颈。

大家也发现**正向传播中间状态占的内存太多了，可以玩一个用算力换内存的把戏**，就是不要存储那么多梯度和每一层的正向传播的中间状态，而是在计算到某一层的时候再临时从头开始重算正向传播的中间状态，这样这层的正向传播中间状态就不用保存了。如果每一层都这么干，那么就只要 2 个字节来存这一层的梯度。但是计算中间状态的算力开销会很大。因此实际中一般是把整个 Transformer 分成若干组，一组有若干层，只保存每组第一层的中间状态，后面的层就从该组第一层开始重新计算，这样就平衡了算力和内存的开销。

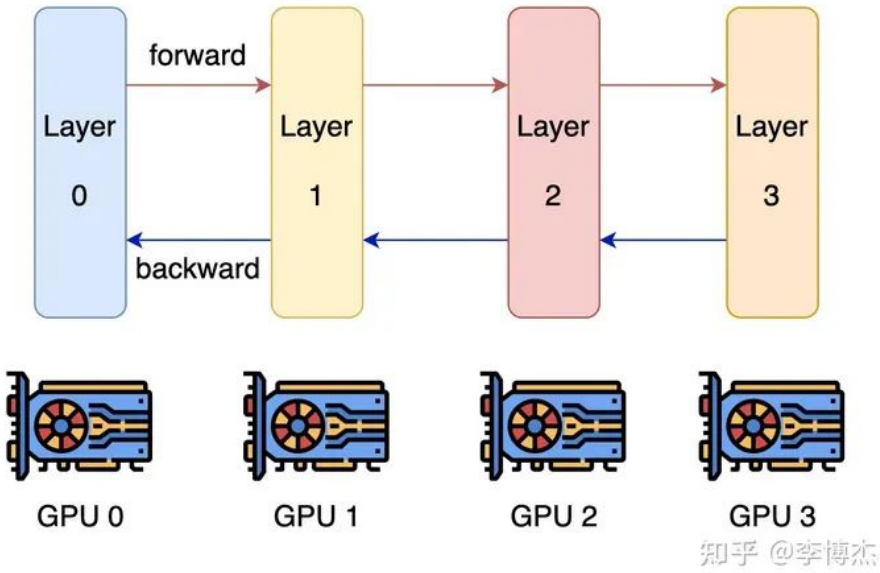
如果还是算不清楚，可以读读这篇论文：Reducing Activation Recomputation in Large Transformer Models。

当然有人说，GPU 内存放不下可以换出到 CPU 内存，但是就目前的 PCIe 速度，换出到 CPU 内存的代价有时候还不如在 GPU 内存里重算。如果是像 Grace Hopper 那种极高带宽的统一内存，那么换入换出倒是一个不错的主意，不管训练的正向传播中间状态还是 KV Cache，都有很多优化的空间。

Pipeline parallelism（流水线并行）

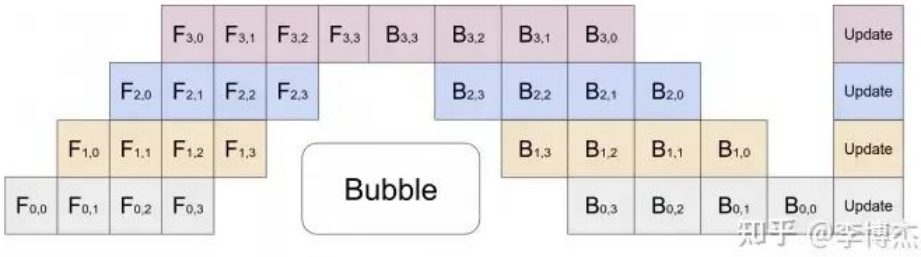
既然一块 GPU 放不下，用多块 GPU 总行了吧？这就是 **model parallelism（模型并行），可以大致分为 pipeline parallelism 和 tensor parallelism。**

大家最容易想到的并行方式就是 **pipeline parallelism**，模型不是有很多层吗，那就分成几组，每组算连续的几层，穿成一条链。



Pipeline Parallelism 示意图，来源：Colossal AI

这样就有个问题，**一条链上只有一个 GPU 在干活，剩下的都在干等。**当然聪明的你一定也想到了，既然叫 pipeline，那就可以流水线处理，可以把一个 batch 分为若干个 mini-batch，每个 mini-batch 分别计算。



Pipeline Parallelism 示意图，来源：GPipe

这可好，是不是把 pipeline 搞的越深越好，每个 GPU 只算一层?

首先，正向传播中间状态（activation）的存储容量会成倍增加，加剧内存容量不足的问题。比如流水线的第一级算出了正向传播的中间状态，如果有 N 个流水级，那就要正向流过后面的 N - 1 个流水级，再等反向传播 N - 1 个流水级，也就是 2N - 2 轮之后才能用到这个正向传播的中间状态。不要忘了每一轮都会产生这么多中间状态，因此一共是保存了 2N - 1 个中间状态。如果 N 比较大，这个存储容量是非常恐怖的。

其次，pipeline 的相邻流水级（pipeline stage）之间是要通信的，级数越多，通信的总数据量和总时延就越高。

最后，要让这样的 pipeline 流起来，batch size 需要等于 Transformer 里面的层数，一般是几十，再乘以 data parallelism 的并行数，batch size 会很大，影响模型收敛的速度或模型收敛后的精度。

因此，**在内存容量足够的情况下，最好还是少划分一些流水级。**

对于 LLaMA-2 70B 模型，模型参数需要 140 GB，反向传播的梯度需要 140 GB，优化器的状态（如果用 Adam）需要 840 GB。

正向传播的中间状态跟 batch size 和选择性重新计算的配置有关，我们在算力和内存之间取一个折中，那么正向传播的中间状态需要 token 长度 * batch size * hidden layer 的神经元数量 * 层数 * (10 + 24/张量并行度) 字节。假设 batch size = 8，不用张量并行，那么 LLaMA-2 70B 模型的正向传播中间状态需要 4096 * 8 * 8192 * 80 * (10 + 24) byte = 730 GB，是不是很大?

总共需要 140 + 140 + 840 + 730 = 1850 GB，这可比单放模型参数的 140 GB 大多了。一张 A100/H100 卡也只有 80 GB 内存，这就至少要 24 张卡；如果用 4090，一张卡 24 GB 内存，就至少需要 78 张卡。

LLaMA-2 模型一共就只有 80 层，一张卡放一层，是不是正好？这样就有 80 个流水级，单是流水线并行就有 80 个并行的 batch 才能填满流水线。

这样，**正向传播的中间状态存储就会大到无法忍受**，这可是 80 * 2 = 160 轮的中间状态，翻了 160 倍。就算是使用选择性重新计算，比如把 80 层分成 8 组，每组 10 层，中间状态存储仍然是翻了 16 倍。

除非是用最极端的完全重新计算，反向传播到每一层都重新从头开始计算正向传播的中间结果，但这样计算开销可是随模型层数平方级别的增长，第 1 层算 1 层，第 2 层算 2 层，一直到第 80 层算 80 层，一共算了 3240 层，计算开销可是比正常算一次 80 层翻了 40 倍，这还能忍?

中间状态存储的问题就已经够大了，再看这 2048 张卡之间的通信开销。按照一张卡放一层，并且用不同的输入数据让它完全流水起来的做法，这 2048 张卡分别在计算自己的 mini-batch，可以认为是独立参与到 data parallelism 里面了。前面讲过，在数据并行中，每一轮需要传输的是它计算出的梯度和全局平均后的梯度，梯度的数据量就等于模型的参数数量。

把 70B 模型分成 80 层，每一层大约有 1B 参数，由于优化器用的是 32 bit 浮点数，这就需要传输 4 GB 数据。那么一轮计算需要多久呢？总的计算量 = batch size * token 数量 * 6 * 参数量 = 8 * 4096 * 6 * 1B = 196 Tflops，在 4090 上如果假定算力利用率 100%，只需要 0.6 秒。而通过 PCIe Gen4 传输这 4 GB 数据就已经至少需要 0.12 秒了，还需要传两遍，也就是先传梯度，再把平均梯度传过来，这 0.24 秒的时间相比 0.6 秒来说，是占了比较大的比例。

当然我们也可以做个优化，让每个 GPU 在 pipeline parallelism 中处理的 80 组梯度数据首先在内部做个聚合，这样理论上一个 training step 就需要 48 秒，通信占用的时间不到 1 秒，通信开销就可以接受了。当然，通信占用时间不到 1 秒的前提是机器上插了足够多的网卡，能够把 PCIe Gen4 的带宽都通过网络吐出去，否则网卡就成了瓶颈。假如一台机器上插了 8 块 GPU，这基本上需要 8 块 ConnectX-6 200 Gbps RDMA 网卡才能满足我们的需求。

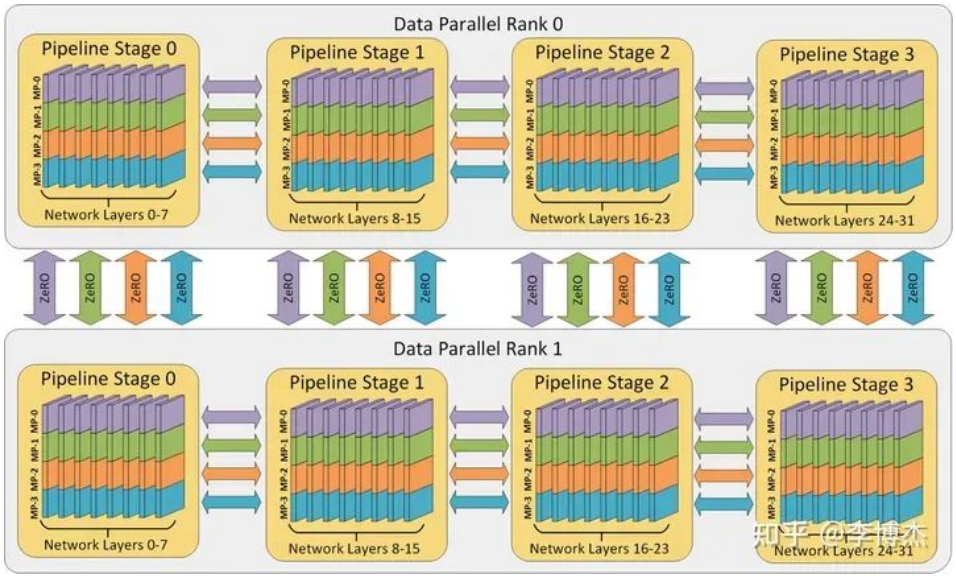
最后再看 batch size，整个 2048 张卡的集群跑起来，每个 GPU 的 mini-batch 我们刚才设置为 8，那可真是 batch size = 16384，已经是大规模训练中比较大的 batch size 了，如果再大，可能就影响模型的收敛速度或收敛后的精度了。

因此，**单纯使用流水线并行和数据并行训练大模型的最大问题在于流水线并行级数过多，导致正向传播中间状态（activation）存储容量不足。**

Tensor parallelism（张量并行）

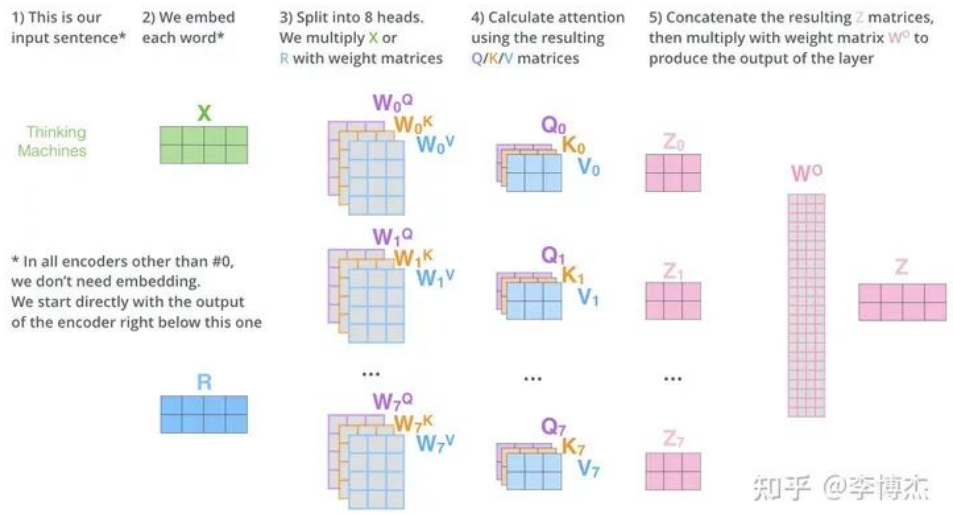
那就没办法了吗？我们还有最后一招，就是 Tensor parallelism（张量并行）。它也是模型并行的一种，但不像流水线并行那样是在模型的层间划分，而是在模型的层内划分，也就是把一层内的 attention 计算和 Feed Forward Network 划分到多个 GPU 上处理。

有了张量并行，就可以缓解 GPU 放不下模型导致的流水级太多的问题。分到 80 个 GPU 才能放下的模型，如果用单机 8 卡张量并行，就只需要划分 10 个流水级。同时，张量并行还可以降低 batch size，因为张量并行的几个 GPU 是在算同一个输入数据。



Tensor、Pipeline、Data 三种并行方式从模型层内、模型层间、训练数据三个维度上划分计算空间，来源：DeepSpeed

Attention 的计算过程是比较容易并行的，因为有多个 head，用来关注输入序列中的不同位置的，那么把这些 head 分别拆开就行了。



Attention 的计算过程，来源：The Illustrated Transformer

但是我们做任何并行计算的时候都不要忘记通信开销。

每个 head 里面的 Q、K 两个矩阵的大小是 batch size * token 长度 * key 的大小，V 矩阵的大小是 batch size * token 长度 * value 的大小。key/value 的大小一般等于 embedding size / heads 数量，例如在 LLaMA-2 70B 中就是 8192 / 64 = 128，矩阵大小是 batch size * 4096 * 8192 / 64（注意，这只是一个 head 的）。而 Q、K、V 参数矩阵在每个 head 上的大小是 embedding size * embedding size / heads num = 8192 * 8192 / 64。

我们前面推导过，正向的计算量基本上就是每个 token 过一遍所有参数的计算量，2 * 3 (Q, K, V) * batch size * token 长度 * 参数个数 = 2 * 3 * batch size * 4096 * 8192 * 8192 / 64。可以跟矩阵的大小对一下，看看有没有算错。

那么通信量是多少呢？输出矩阵 Z 是由每个 head 拼起来的，每个 head 的大小是 batch size * token 长度 * embedding size / heads num = batch size * 4096 * 8192 / 64。输入矩阵 X 的大小是 batch size * token 长度 * embedding size = batch size * 4096 * 8192。注意这里的 X 大小跟所有 heads 合并在一起后的 Z 大小是一致的，而我们在这里算的是每个 head 的 Z 大小。这里的单位是参数数量，如果按照字节算，还要乘以每个参数的大小。

如果我们采用最极端的方式，每个 head 交给一个 GPU 去算，那么计算量和通信量的比例是多少？大概是 2 * 3 * embedding size / heads num / bytes per param = 2 * 3 * 8192 / 64 / 2

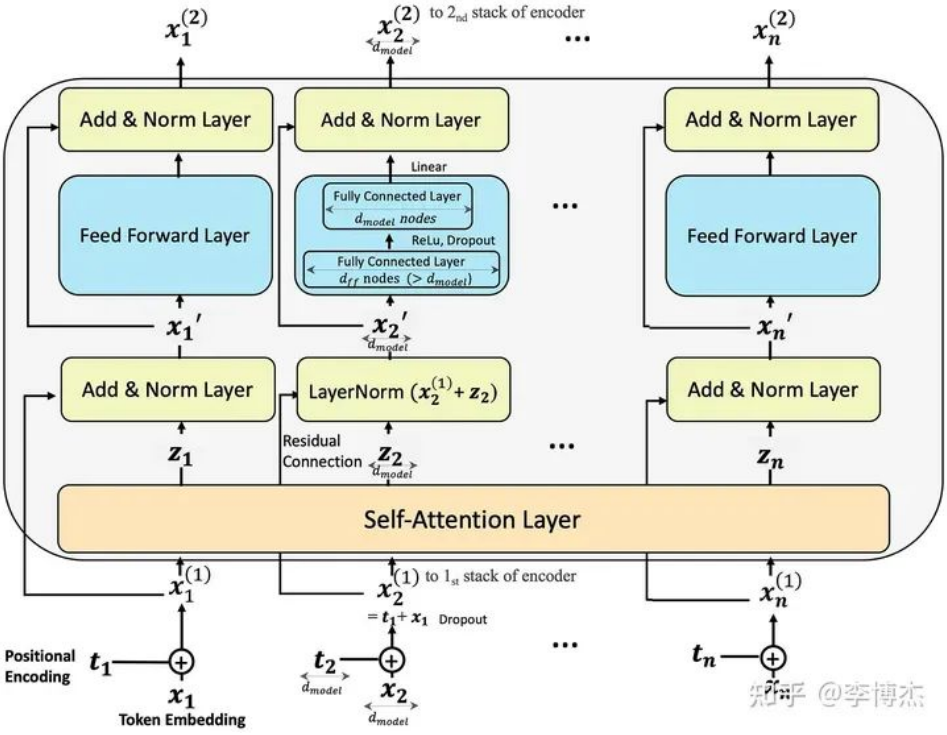
= 384。代入 4090 的 330 Tflops，如果想让通信不成为瓶颈，那么通信带宽至少需要是 330T / 384 = 859 GB/s，发送接收双向还得乘以 2，就是 1.7 TB/s。太大了，远远超过 PCIe Gen4 x16 的 64 GB/s，就算 NVLink 的 900 GB/s 都撑不住。

所以，**tensor parallelism 不能切得太细，每个 GPU 需要多算几个 heads**。如果每个 GPU 多算几个 attention heads，输入矩阵 X 就是这些 heads 共享的了，因此输入矩阵的通信开销就被多个 heads 平摊了，计算量和通信量的比例就可以提高。

还是按照 4090 的算力 / 单向通信带宽 = 330T / (64GB/s / 2) 来算，计算量和通信量的比例最少需要是 10000，也就是 2 * 3 * (embedding size / 张量并行 GPU 数量) / bytes per param = 2 * 3 * 8192 / 张量并行 GPU 数量 / 2 >= 10000，解得：张量并行 GPU 数量 <= 2.4。也就是告诉你，要是用了张量并行，最多用 2 个 GPU，如果用更多的 GPU，算力就肯定跑不满理论值。这让我怎么玩?

但是，如果把 H100 的参数代入进去，马上就不一样了。H100 的峰值算力是 1979 Tflops，NVLink 双向带宽是 900 GB/s，计算量和通信量的比例最少需要是 4400，也就是 2 * 3 * (embedding size / 张量并行 GPU 数量) / bytes per param = 2 * 3 * 8192 / 张量并行 GPU 数量 / 2 >= 4400，解得：张量并行 GPU 数量 <= 5.5，也就是单机 8 卡做张量并行，如果算力跑满，网络会成为瓶颈。可以看到，即使对于 900 GB/s 这么快的 NVLink，在巨大的算力面前，都容易出现茶壶里煮饺子倒不出来的情况。当然，采用更优的并行切分方式可以节约一些网络通信开销。

阉割版的 H800 相比 H100 卡的就是网络带宽，把网络带宽从 900 GB/s 降到 400 GB/s 了。我们再代入一次，计算量和通信量比例最少需要是 10000，那么张量并行 GPU 数量 <= 2.4，跟 4090 一个货色了。这样单机 8 卡做张量并行，就会导致网络成为瓶颈。当然，计算量 1979 Tflops 是理论值，并行切分方式也可以优化，因此实际训练 70B 的模型 8 卡 H800 网络不一定真的是瓶颈。**这就是 H800 精准打击大模型训练，让张量并行过得不舒服。**



Feed Forward Network 的计算过程，虽然这是 encoder 的，但 decoder 也差不多，来源：Step-by-Step Illustrated Explanations of Transformer

如果在 Feed Forward Network 这里做张量并行，也是可以做类似的推导，在这里就不赘述了。大凡神经网络里的矩阵乘法，M*N 的矩阵乘上 N*K 的矩阵，总的计算量是 M*N*K，输入输出的总大小是 (M*N + N*K)，多摞几个矩阵那也是常数（就像 Q、K、V），也就是计算和通信的比例跟矩阵的边长（dimension）是一个量级的。

这么分析完了，**如果你是要做大规模大模型训练，你还会买 A100/H100/H800 的 PCIe 版吗?** PCIe Gen5 虽然比 Gen 4 快一倍，但对 H100 而言，计算量和通信量的比例仍然最少需要

是 1979T / (128G / 2) = 30000，解出来张量并行 GPU 数量 <= 0.8，只要用了张量并行，就是损失算力的！

等到 H100 的下一代出来了，比如 GH200，算力又翻了一倍，NVLink 还是 900 GB/s，这时候 NVLink 就也开始有点吃力了。所以 GH200 不失时机的推出了统一大内存，号称 144 TB，就是为了更好的做换入换出，用内存换网络通信。如果禁令保持不变，国内版本还是卡住 400 GB/s 的通信，那性能差距会有多大？

上面的推导当然都是简化的，实际上可能不会这么夸张，但数量级是差不多的。

训练部分小结

4090 不容易做大模型训练的原因除了前面分析的**内存小，通信慢，license 不支持数据中心**，还有很多其他问题。

比如，A100/H100 支持 ECC 显存容错，据说 4090 也支持 ECC，但是不知道故障率会不会比 A100/H100 更高。不要小看了容错，2048 张卡的集群就算每张卡 1 个月出一次故障，平均 20 分钟就会有一张卡出故障！要是没有自动化的故障恢复方式，炼丹师就别想睡觉了。

就算是自动从上一个 checkpoint 恢复，这可是要时间的，如果不考虑丢弃故障 GPU 梯度这种比较暴力的方式，当前这个 step 就算是白算了，还要从上一个 checkpoint 加载梯度，一般需要 10 来分钟的时间才能搞定。这样，每 20 分钟就浪费 10 分钟，这 10 分钟恢复过程中可能又有新的卡故障，总的算下来要浪费掉一半的有效算力。

因此，**保持大规模训练集群的低故障率是非常重要的**，这些 GPU 卡都非常金贵，可不能像挖矿机房那样，动不动就过热死机了。

据说 3090 是支持 NVLink 的，但 4090 就把 NVLink 给砍掉了。更老的卡，甚至还有支持 PCIe P2P 的，现在也都被砍掉了。谁感兴趣可以测一测 3090 的 NVLink 性能怎么样，是不是真的能达到标称的 600 GB/s，如果真的能达到的话，是否又可以用来做大模型训练了呢。

我们年会的时候，海哥讲了个段子，我们找老婆都希望又漂亮，又能挣钱，还一心一意爱自己。可同时满足这三个条件的老婆就很难找到了。类似的，在分布式系统中，我们都希望性能又高，通用性又强，成本还低。这三个条件的交集也很小。海哥讲到这里，谭博补充了一句，同时满足这三个条件的分布式系统根本就不存在。

Tensor、Pipeline、Data Parallelism 就像是这样的不可能三角，相互牵制，只要集群规模够大，模型结构仍然是 Transformer，就很难逃出内存容量和网络带宽的魔爪。

大模型推理为什么 4090 很香

推理和训练有什么区别？

首先，训练不仅需要存储模型参数，还需要存储梯度、优化器状态、正向传播每一层的中间状态（activation），后面几个比参数更大，对模型内存的需求量也更大。

其次，训练任务是一个整体，流水线并行的正向传播中间结果是需要存下来给反向传播用的。为了节约内存而使用流水线并行，流水级越多，要存储的中间状态也就更多，反而加剧内存的不足。而推理任务中的各个输入数据之间并没有关系，正向传播每一层的中间状态也不需要保存下来，因此流水线并行不需要存储很多中间状态。

首先我们需要计算一下推理需要多少算力。前面针对训练算力的估算，为了简单起见，忽略了两个事情，首先是没有考虑 KV Cache，其次是没有考虑内存带宽。

KV Cache

什么是 KV Cache？对于每个输入的 prompt，在计算第一个 token 输出的时候，每个 token 的 attention 肯定是都要从头计算。但是在后续 token 的生成中，都需要计算 self-attention，也就是输入 prompt 以及前面输出的 token 的 attention。这是就需要用到前面每一个 token 的 K 和 V，由于每一层的参数矩阵是不变的，此时只有刚生成的那个 token 的 K 和 V 需要从头计算，输入 prompt 和之前生成的 token 的 K 和 V 其实是跟上一轮一样的。

这时，我们就可以**把每一层的 K、V 矩阵缓存起来，生成下一个 token 的时候不再需要重新计算，这就是所谓的 KV Cache。**Q 矩阵每次都不一样，没有缓存的价值。前面讲的训练中的选择性保存正向 activation 是个拿计算换内存的把戏，这里的 KV Cache 就是一个拿内存换计算的把戏。

KV Cache 需要多少存储容量呢？每一层，每个 token 的 K、V 矩阵都是 embedding size 这么大，再乘上 token 数量和 batch size，就是这一层的 KV Cache 所需的存储容量了。一定要记住 batch size，在正向和反向传播的几乎所有阶段，都不会涉及到对 batch size 中各个 sample 的合并处理，因此它始终是存储量和计算量计算中的一个系数。

例如，如果 batch size = 4，在 LLaMA 2 70B 中，假设输入和输出的 token 数量达到了模型的极限 4096，80 层的 KV Cache 一共需要 2 (K, V) * 80 * 8192 * 4096 * 8 * 2B = 80 GB。如果 batch size 更大，那么 KV Cache 占据的空间将超过参数本身占的 140 GB。

KV Cache 能省下来多少计算量？每一层计算 K、V 矩阵一共需要 2 (K, V) * 2 (mult, add) * embedding size * embedding size = 4 * 8192 * 8192 这么多计算量，乘以之前输入过的 token 数量、层数和 batch size，就是 4096 * 80 * 8 * 4 * 8192 * 8192 = 640 Tflops。相当于每存储 1 个字节，节约了 16K 次计算，还是很划算的。

事实上，KV Cache 节约的远远不止这些。计算 K、V 矩阵的过程是个典型的内存密集型过程，它需要加载每一层的 K、V 参数矩阵。也就是如果不做任何缓存，假设 prompt 长度很短而输出长度接近 token 的最大长度 4096，到了最后一个 token 的时候，单是重复计算前面每个 token 的 K、V 矩阵，就需要读取内存 4096 * 80 * 2 * 8192 * 8192 = 40T 次，每次 2 个字节，要知道 H100 的内存带宽只有 3.35 TB/s，4090 更是只有 1 TB/s，这单是最后一个 token 就得耗掉一张卡几十秒的时间来做重复计算。这样，token 的输出就会越来越慢，整个输出时间是输出长度平方级别的，根本没法用。

推理是计算密集还是存储密集

接下来我们就可以计算推理所需的计算量了。总的算力很好算，前面讲过，大概就是 **2 * 输出 token 数量 * 参数数量 flops。**如果想看细节，可以看下面这张图，来源是这里。

Symbol	Definition	Shape	FLOP	I/O	FLOP:I/O
Scalars					
Input Shape					
b	Batch size	1			
s	Sequence length	1			
Model Hyper-parameter					
n	The number of attention heads	1			
d	Hidden state size of one head	1			
h	Hidden state size ($h = n \cdot d$)	1			
Parameters					
W_Q, W_K, W_V	Projection for Q, K, V	(h, h)			
W_O	Projection for self-attention output	(h, h)			
W_1	First layer in the FFN	$(h, 4h)$			
W_2	Second layer in the FFN	$(4h, h)$			
Initial Computation					
Input					
X	Input for self attention	(b, s, h)			
Self-Attention					
$Q^\ddagger, K^\ddagger, V^\ddagger$	XW_Q, XW_K, XW_V	(b, s, h)	$O(bsh^2)$	$O(2bsh + h^2)$	$O(1/(1/h + 1/bs))$
$Q^\dagger, K^\dagger, V^\dagger$	Reshape $Q^\ddagger, K^\ddagger, V^\ddagger$	(b, s, n, d)			
Q, K, V	Transpose $Q^\dagger, K^\dagger, V^\dagger$	(b, n, s, d)			
K^T	Transpose K	(b, n, d, s)			
P	Softmax $(QK^T/\sqrt{d})$	(b, n, s, s)	$O(bs^2nd)$	$O(2bsnd + bs^2n)$	$O(1/(1/d + 1/s))$
$A^\ddagger$	PV	(b, n, s, d)	$O(bs^2nd)$	$O(2bsnd + bs^2n)$	$O(1/(1/d + 1/s))$
$A^\dagger$	Transpose $A^\ddagger$	(b, s, n, d)			
A	Reshape $A^\dagger$	(b, s, h)			
Y	AW_O	(b, s, h)	$O(bsh^2)$	$O(2bsh + h^2)$	$O(1/(1/h + 1/bs))$
Feed-Forward Network					
Z	$\text{ReLU}(YW_1)W_2$	(b, s, h)	$O(16bsh^2)$	$O(2bsh + 8h^2)$	$O(1/(1/h + 1/bs))$
Auto-Regression					
Input					
x	Input for self attention	$(b, 1, h)$			
K_s, V_s	Key/Value cache from past	(b, n, s, d)			
Self-Attention					
$q^\ddagger, k^\ddagger, v^\ddagger$	xW_Q, xW_K, xW_V	$(b, 1, h)$	$O(bh^2)$	$O(2bh + h^2)$	$O(1/(1/h + 1/b))$
$q^\dagger, k^\dagger, v^\dagger$	Reshape $q^\ddagger, k^\ddagger, v^\ddagger$	$(b, 1, n, d)$			
q, k, v	Transpose $q^\dagger, k^\dagger, v^\dagger$	$(b, n, 1, d)$			
K, V	$\text{concat}(K_s, k), \text{concat}(V_s, v)$	$(b, n, s+1, d)$			
K^T	Transpose K	$(b, n, d, s+1)$			
p	Softmax $(qK^T/\sqrt{d})$	$(b, n, 1, s+1)$	$O(bsnd)$	$O(bsn + bsnd + bnd)$	$O(1/(1 + 1/d + 1/s))$
$a^\ddagger$	pV	$(b, n, 1, d)$	$O(bsnd)$	$O(bsn + bsnd + bnd)$	$O(1/(1 + 1/d + 1/s))$
$a^\dagger$	Transpose $a^\ddagger$	$(b, 1, n, d)$			
a	Reshape $a^\dagger$	$(b, 1, h)$			
y	aW_O	$(b, 1, h)$	$O(bh^2)$	$O(2bh + h^2)$	$O(1/(1/h + 1/b))$
Feed-Forward Network					
z	$\text{ReLU}(yW_1)W_2$	$(b, 1, h)$	$O(16bh^2)$	$O(2bh + 8h^2)$	$O(1/(1/h + 1/b))$

Transformer 推理过程中每一步的矩阵形状、所需算力和内存访问量，来源：Lequn Chen，Dissecting Batching Effects in GPT Inference

但算力并不能说明一切，**模型还需要访问 GPU 内存，内存带宽也可能成为瓶颈。**至少需要把参数从内存里面读出来吧？事实上，内存带宽的估算就这么简单，内存访问量 = 参数数量 * 2 bytes。中间结果有一部分是可以放在缓存里面的，缓存放不下的部分也需要占内存带宽，我们先不算。

如果不做任何批量输入，也就是模型专门服务一个 prompt，batch size = 1，整个 context 的长度很短（例如只有 128），那么整个推理过程中，每载入一个参数（2 字节），就只进行 128 次乘法和加法计算，那么计算 flops 和访问内存 bytes 的比例就只有 128。基本上任何 GPU 在这种情况下都会变成 memory bound，时间都耗在加载内存上了。

对于 4090 来说，计算 flops 和内存带宽之比是 330 / 1 = 330；对于 H100 来说，计算 flops 和内存带宽之比是 1979 / 3.35 = 590。也就是说，如果 context 中的 token 数量小于 330 或者 590，那么内存访问就会成为瓶颈。

虽然 LLaMA 2 的理论上限是 4096 个 token，但很多输入 prompt 用不了这么多，因此内存访问是有可能成为瓶颈的。此时，就需要靠 batch size 来补足了。推理中的批量处理，就是把几乎同时到达后端服务的 prompt 放到一起处理。不用担心，batch 里面的不同 prompt 的处理是完全独立的，不用担心会互相干扰。但这些 prompt 的输出是步调整齐划一的，每一轮整个 batch 中的每个 prompt 都会输出一个 token，因此如果有的 prompt 先输出完了，那就只能等其他的输出结束，造成一定的算力浪费。

有的人问，批量处理所需的算力跟分别单独处理所需的算力是一样的呀，那推理时为什么需要批量处理？答案就在访问内存的带宽上。

如果同时到达服务器的 prompt 很多，是不是 batch size 越大越好？也不是，因为 KV Cache 的大小可是正比于 batch size 的，batch size 大了，KV Cache 占据的 GPU 内存容量就很可

观，比如在 LLaMA-2 70B 中，每个 prompt 都要占据 5 GB 的 KV Cache，如果 batch size 搞到 32，那么 KV Cache 就会占掉 160 GB 的 GPU 内存，比参数都大了。

70B 推理需要多少张卡?

总的存储容量也很好算，推理的时候最主要占内存的就是参数、KV Cache 和当前层的中间结果。当 batch size = 8 时，中间结果所需的大小是 batch size * token length * embedding size = 8 * 4096 * 8192 * 2B = 0.5 GB，相对来说是很小的。

70B 模型的参数是 140 GB，不管 A100/H100 还是 4090 都是单卡放不下的。那么 2 张 H100 够吗？看起来 160 GB 是够了，但是剩下的 20 GB 如果用来放 KV Cache，要么把 batch size 压缩一半，要么把 token 最大长度压缩一半，听起来是不太明智。因此，至少需要 3 张 H100。

对于 4090，140 GB 参数 + 40 GB KV Cache = 180 GB，每张卡 24 GB，8 张卡刚好可以放下。

推理用流水线并行可以吗?

推理使用流水线并行，最主要的问题是串行处理的推理延迟，网络延迟倒是小问题。

首先是推理延迟。虽然流水线的不同阶段可以塞进不同的 prompt，但同一个 prompt 的处理仍然永远在单个 GPU 上轮转，这样相比 Tensor parallelism 而言，单个 prompt 的延迟就增大了。

对于很小的 batch size，GPU 内存带宽是瓶颈，此时每张卡计算每个 token 的时延就是 2 byte * 参数量 / 卡的数量 / 内存带宽，例如 8 卡 4090 跑 LLaMA-2 70B，就是 2 * 70G / 8 / 1 TB/s = 0.0175 秒。这里没有考虑 KV Cache 带来的节约。注意，8 张卡是串行处理的，因此每个 token 的时延还要乘以 8，也就是 0.14 秒。**每秒只能输出 7 个 token**，对于 70B 这么小的模型来说是有点慢了。

对于很大的 batch size，GPU 算力是瓶颈，此时每张卡计算每个 token 的时延就是 batch size * 2 * 参数量 / 卡的数量 / 算力，例如 batch size = 1024，同样的 8 卡例子，就是 1024 * 2 * 70G / 8 / 330 Tflops = 0.0543 秒。事实上，对于这么大的 batch size，KV Cache 和正向传播的中间结果先把 GPU 内存给吃满了。

那么**要平衡利用 GPU 算力和内存带宽，batch size 需要是多少呢?** 这就是 2 byte * 参数量 / 卡的数量 / 内存带宽 = batch size * 2 * 参数量 / 卡的数量 / 算力，左右两边参数量和卡的数量互相抵消，得到 batch size = 算力 / 内存带宽。对于 4090，就是 330 / 1 = 330；对于 H100，就是 1979 / 3.35 = 590。也就是说，对 4090 而言，batch size 小于 330 的时候 GPU 内存带宽是瓶颈，大于 330 的时候 GPU 算力是瓶颈。当 batch size = 330 的时候，理想情况下，内存带宽和算力恰好都打满，每张卡处理每个 token 的时间就是 17.5 ms。

其次是**网络延迟**。流水线并行相比张量并行的优点就是网络传输量小，流水级之间只需要传输 batch size * embedding size 这么多数据。例如 batch size = 8，embedding size = 8192，只需要传输 128 KB 数据，在 32 GB/s 的 PCIe Gen4 x16 上，只需要 4 us 就可以传输完成。当然，还需要考虑到通信库本身的开销，加上 4090 不支持 GPU 之间 P2P 传输，需要通过 CPU 中转，实际上需要几十 us 的时间，相比计算部分动辄几十 ms 的时延，可以忽略不计。

即使 batch size = 330，这 5.28 MB 数据在 PCIe 上也只需要传输 0.16 ms，相比计算部分的 17.5 ms 仍然可以忽略不计。

如果可以忍受流水线并行的推理延迟，甚至可以用多台主机来做流水线并行。我们假设主机间只有 1 Gbps 的普通以太网络，每台主机只有一张 4090。对于 batch size = 1，16 KB 数据需要 0.25 ms 才能传输完成，再加上 0.25 ms 两端网络协议栈的处理时间，每个流水级就需要 0.5 ms 的时延，8 张卡花在通信上的时间只有 4 ms，相比整体计算时延 140 ms 来说可以忽略，不会显著影响系统的推理延迟。

当 batch size 很小时，流水线推理中的网络流量是突发性（bursty）的，每过 18 ms 只会进行 0.25 ms 数据传输，只有 1/72 的占空比，不用担心流水线推理把局域网全部给占满了，搞得没法正常上网了。

如果为了充分利用算力，把 batch size 设置得很大，比如 330，那么 16 KB * 330 = 5.28 MB 数据需要传输 41 ms，8 张卡花在通信上的时间高达 0.33 秒，这样就只有 3 token/s 的输出速度了，难以忍受。因此，**如果用主机间通信来做流水线并行，主机间又没有很高的通信带宽，就势必需要牺牲一定的吞吐量。**

例如，我们设置输出速度不小于 5 token/s，这时留给通信的时间是 60 ms，每个流水级至多 7.5 ms，1 Gbps 网络可以传输 960 KB 数据，这时 batch size 至多设置为 60，也就是这 8 张 4090 的总吞吐量是 2400 token/s。此时的有效算力利用率只有不到 20%。

最近有一个比较火的 Petals 开源项目，就是利用流水线并行，把 GPU 做成了一个类似 BitTorrent 的分布式网络。虽然推理延迟确实比较高，但至少说明了分布式 GPU 推理的可行性。

推理用张量并行怎么样？

前面讲到，流水线并行的最大缺点是 GPU 串行处理，延迟较高，导致输出 token 比较慢。而张量并行的最大缺点是传输数据量大，网络带宽低的设备不一定 hold 得住。

但是推理要传输的数据量跟训练要传输的数据量可不是一回事啊！推理只需要传输正向传播的中间结果（activation），而训练还需要传输所有参数的梯度，梯度才是数据量的大头。

在推理中，如果使用张量并行，Transformer 的每一层都需要传输把自己负责的结果向量（大小为 batch size * embedding size / num GPUs）广播给其他所有 GPU，并接受来自所有其他 GPU 广播来的数据。计算 attention 的时候需要传输一次，计算 feed-forward network 的时候又需要传输一次，也就是总共需要传输 2 * 层数这么多次。

每次发送就是 batch size * embedding size（发送和接收是不同的方向，不能算两次），对于 batch size = 1, embedding size = 8192，只需要传输 16 KB 数据，在 32 GB/s 的 PCIe Gen4 上传输只需要 1 us。当然，考虑到前面讨论的 CPU 中转开销，还是需要大约 30 us 的。一共 160 次传输，需要 4.8 ms。

我们再考虑计算的开销。还是考虑 batch size = 1 的情形，GPU 内存带宽是瓶颈，此时每张卡计算每个 token 的时延就是 2 byte * 参数量 / 卡的数量 / 内存带宽，代入我们前面的数值，仍然是 17.5 ms。但是这里 8 张卡是并行处理的，因此总的处理时长就是计算时间 + 通信时间 = 17.5 ms + 4.8 ms = 22.3 ms。这就意味着**每秒可以生成 45 个 token**，这个 token 生成速度已经很不错了，至少人类的阅读速度是很难赶上生成的速度了。

如果 batch size 更大会怎样？例如 batch size = 330，把 GPU 算力和内存带宽都充分利用起来，每次需要传输的数据量是 330 * 8192 * 2 = 5.4 MB，在 32 GB/s 的 PCIe Gen4 上需要 0.17 ms。一共 160 次传输，就是 27 ms。这下网络通信开销成了延迟的大头，总处理时长为 27 + 17.5 = 44.5 ms，**每秒只能生成 22 个 token 了**，但也不算慢。

注意，不管用多少个 GPU 做并行推理，只要用的是张量并行，网络传输的总数据量是相同的，因此**增加 GPU 的数量只能加速计算，不能加速通信。**

因此，**A100/H100 的 NVLink 在降低推理延迟方面还是有很大作用的。**如果用 A100/H100，取 batch size = 590 达到算力和带宽的平衡利用，这 9.44 MB 数据只需要 9.44 MB / 450 GB/s = 0.02 ms。一共 160 次传输，也只有 3.2 ms。由于内存带宽大了，计算时间也可以大幅缩短，例如 H100 的计算时间为 2 * 70G / 8 / 3.35 TB/s = 5.2 ms。总处理时长只有 5.2 ms + 3.2 ms = 8.4 ms，**每秒可以生成 119 个 token**，非常棒！

可以说，如果论单个 prompt 的 token 生成速度，无论用多少块 4090 也追不上 8 卡 H100。

用 4090 做推理的成本怎么样？

对于推理，不管用流水线并行还是张量并行，batch size 不算高到太离谱的情况下内存带宽都是瓶颈。

假如 batch size 能够高到把算力 100% 利用起来，并且还能解决 KV Cache 不够大的问题，能解决中间结果占用内存过多的问题，那么这 8 张 4090 可以达到多少吞吐量?

当然，这两个问题都不好解决，因此**推理优化才是一个热门的研究领域，存在很多的 trade-off 和奇技淫巧。**如果只是用标准的 PyTorch，那推理性能距离把算力 100% 利用起来还远得很呐。

假设都解决了，在张量并行的通信过程中我们可以利用 double buffer 做另外一个 batch 的计算，也就是计算和通信并行，进一步提高吞吐量。通信和计算分别是 27 ms 和 17.5 ms，传输的 27 ms 是瓶颈，也就是每 27 ms 输出一组 token，一个 batch 330 个 prompt，那这 8 张 4090 真是可以达到每秒 330 / 0.027 = 12.2K token 的吞吐量。

8 张 4090 的成本是 12800 美金，8 卡 PCIe Gen4 服务器本身要 2 万美金，加上网络设备，平均每台 4 万美金的设备成本。固定资产按照 3 年摊销，每小时 1.52 美元。整机功耗大约 400W * 8 + 2 kW = 5 kW，按照 0.1 美元一度电算，每小时 0.5 美元。这 2 美元一小时的机器，满打满算能生成 12.2K * 3600 = 44M tokens，也就是说 **1 美元能生成 22M tokens。**

是不是比 GPT-3.5 Turbo 的 $0.002 / 1K tokens，也就是 1 美元 0.5M tokens **便宜 44 倍**？当然，账不能这么算。

首先，GPU 的算力利用率到不了 100%；
其次，如同所有 SaaS 服务一样，用户的请求数量有波峰有波谷，用户是按量付费的，平台提供方可是不管有没有人用都在烧钱的；
此外，每个 batch 中不同 prompt 的长度和响应 token 数量都不同，消耗的算力是 batch 中最大的那个，但收的钱是用户实际用的 token 数；
再次，GPT-3.5 是 175B 的模型，比 70B 的 LLaMA 很可能推理成本更高；
最后，OpenAI 开发 GPT-3.5 是烧了不知道多少钱的，人家至少要赚回训练成本和研发人员的工资吧。

其实 GPT-3.5 Turbo 的 $0.002 / 1K tokens 真的挺良心的，有的卖 API 的，LLaMA-2 70B 都敢比 GPT-3.5 Turbo 卖得贵。

如果换成用 H100 做推理，重新算一下这笔账。一张 H100 至少要 3 万美金，一台 8 卡 H100 高配服务器加上配套的 IB 网络，起码要 30 万美金，同样按照 3 年摊销，每小时 11.4 美元。10 kW 功耗，电费每小时 1 美元。**一共 12.4 美元一小时。**

这其实已经是非常良心的价格了，你在任何云服务商都不可能租得到这么便宜的 8 卡 H100。所以说从云服务商租卡卖没有护城河的 SaaS 服务，比如开源模型的推理 API，除非有一种提高推理性能的独门绝技，很难赚得了什么大钱，二房东的生意不是这么好做的。

再算算这台 8 卡 H100 机器的吞吐量，张量并行也采用传输和计算并行，H100 的通信比较快，因此计算是瓶颈，每 5.2 ms 可以输出一组 token，一个 batch 590 个 prompt，满打满算可以达到每秒 590 / 0.0052 = 113K token 的吞吐量。理想情况下，一小时能生成 407M tokens，也就是 1 美元能生成 33M tokens，**H100 这单位 token 的成本比 4090 还要低 30%。**

为什么 8 卡 H100 机器是 4090 机器价格的 6 倍，性价比却比 4090 高? 因为一张 H100 的算力是 4090 的 6 倍，内存带宽是 4090 的 3.35 倍，当 batch size 够大，算力达到瓶颈的时候，单卡的性能就是 6 倍。而且，H100 比 4090 的网络带宽强太多了，导致 4090 在张量并行中网络通信成了瓶颈，浪费了有效算力。因此，同样的 8 卡机器吞吐量几乎可以达到 4090 的 10 倍。虽然一张 H100 卡的价格是 4090 的 20 倍以上，但算上服务器本身的成本和电费，整机的成本只是 6 倍左右。

用最便宜的设备搞出最高的推理性能

我们发现在 8 卡 4090 机器中，3 万美金的设备成本，GPU 卡只占了 1.28 万美金，不像 H100 机器那样 GPU 成本占了大头。还有办法进一步降低吗?

如果我们可以忍受 5 token/s 的输出速度，甚至**可以利用流水线并行，用家用台式机和 4090 攒出个推理集群来。**

遥想我当年在 MSRA 的时候，在一台只用 1000 美金攒出来的机器上插了 10 块 FPGA，做出个世界最快的 Key-Value Store。其实如果让我去设计一个性价比最高的 4090 推理集群，有很多种方案可以尝试：

用流水线并行，台式机 + 10 Gbps 网卡，足够在 5 ms 内传输 batch size = 330 的 5.28 MB 数据了，通信 40 ms，计算 140 ms，达到 5 token/s 的单 prompt 输出速度，同时又能充分利用 4090 的算力。10 Gbps 的网卡和交换机都很便宜，Intel X710 网卡只要 150 美金，20 口交换机只要 1500 美金（每 8 个口 750 美金），一台家用台式机 700 美金，这只要 2 万美金就可以搞定原本需要 4 万美金的设备。
用张量并行，台式机 + 200 Gbps ConnectX-6 网卡，上 RoCE，可以把 batch size = 330 的 5.28 MB 数据在 0.22 ms 内传完，160 次传输是 35 ms，加上计算的 17.5 ms，一个 token 52.5 ms，可以达到 19 token/s 的单 prompt 输出速度，这个速度已经不错了。网卡 1000 美金，200G 交换机 2 万美金 40 个端口，平均每 8 个端口 4000 美金，一台家用台式机 700 美金，这只要 3 万美金就能搞定原本 4 万美金的设备。
主机内用张量并行，主机间用流水线并行，4 卡 PCIe Gen4 服务器主板只要 1000 美金而且能跑满 PCIe 带宽（因为 8 卡就需要 PCIe switch 了，价格会贵很多），两台主机之间用 25 Gbps 网卡直连，主机内张量并行的时延是 27 ms，主机间流水线并行只需 2 次 8 ms 的传输（注意 25G 的网络带宽是 4 张 GPU 卡共享的），加上两次流水线计算各 17.5 ms，总共 78 ms，可以达到 13 token/s 的单 prompt 输出速度。网卡 300 美金 * 2，服务器 3000 美金 * 2，这只要 1.95 万美金就可以搞定原本需要 4 万美金的设备。

2 万美金按照 3 年摊销是每小时 0.76 美元。按照 0.1 美元/度的电价，每小时的电费都要 0.5 美元，接近设备成本了，这有点挖矿的味道了。这 1.26 美元一小时的机器如果跑满了 44M tokens 的吞吐量，1 美元能生成 35M tokens，终于赶上 8 卡 H100 的 33M token per dollar 了。

为什么 H100 以 20 倍于 4090 的 GPU 价格，9 倍的性能，却仍然能在系统性价比上打个平手，首先是因为能耗成本更低，8 卡 H100 的功耗是 10 kW，但 9 台 8 卡 4090 的功耗是 45 kW；其次是因为主机和网络设备成本更低，一台 8 卡 H100 准系统虽然贵，但只占整机价格的 20% 左右；但 4090 因为卡多，除非像 GPU 矿机那样压成本，只要还是用数据中心级的设备，准系统价格就要占到 35% 以上。

其实，这个世界上不止有 A100/H100 和 4090，还有 A10 等计算卡和 3090 等游戏卡，还有 AMD 的 GPU 和很多其他厂商的 AI 芯片。**H100 和 4090 大概率都不是性价比的最优解，例如 A10 和 AMD GPU 的性价比有可能就更高。**

我都想搞一个推理性价比挑战赛，看谁能用最便宜的设备搞出最强的推理吞吐量，同时延迟不能太高；或者用最便宜的设备搞出最低的推理延迟，同时吞吐量不能太低。

这一切都是在假设使用 LLaMA-2 70B 模型，没有做量化压缩的前提下。如果做了量化压缩，那性能就更高，甚至在 Unified Memory 够大的 MacBook Pro 上都能单机跑了。

License 问题怎么办?

我把这个问题放到最后。NVIDIA Geforce driver 的 License 里写道：

> No Datacenter Deployment. The SOFTWARE is not licensed for datacenter deployment, except that blockchain processing in a datacenter is permitted.

既然机器都是用台式机攒起来的，这能叫 data center 吗？还是叫矿场比较合适吧。人家也说了，4090 用来做区块链是允许的。

我有一个大胆的想法，**如果未来的区块链不再用挖矿来做 proof of work，而是用大模型推理来做 proof of work，这是不是很有意思?** 每个人买几块显卡，接到矿池上，既可以自己用来玩游戏，闲时又可以贡献算力。矿池直接就是个卖大模型推理 SaaS 服务的公司，提供前所未有的低

价 API。甚至需要大模型推理服务的人可以在区块链里自己 P2P 玩起来，谁要用大模型就付点 gas。

当然，目前的 proof of work 都是计算很复杂，验证很简单的。如果真用大模型推理做 proof of work，必须防止用户随意编造一个结果交上去。当然这也是有解决方案的，就像 BitTorrent 和其他一些去中心化网络一样，采用信用机制，新人只能做验证别人计算结果的工作，积攒信用；老人每次算错了，都有比较严厉的惩罚。

从另一个角度看，**家庭局域网络的速度也越来越快**，比如我家就自己部署了 10 Gbps 的网络。家中的智能设备越来越多，算力越来越强。光纤入户也越来越普遍，小区和城市的运营商机房里部署了越来越多的边缘计算节点。前面我们用 1 Gbps 的网络就足以把多台主机上的 GPU 组成流水线并行，那么在未来的家庭高速网络中，流水线并行甚至张量并行都将成为可能。

大多数搞 AI 推理的都只关心数据中心，忽略了家中的分布式算力。**只要解决了安全、隐私和经济动机问题，我家的 Siri，也许就跑在邻居家里的 GPU 上。**

很多人都在说要 democratize AI。但现在大模型平民化的最大障碍就是成本，而成本最大的来源又是 GPU 市场上计算卡和游戏卡价格的剪刀差。这并不是指责某家公司，其他做 AI 芯片的公司，AI 芯片的算力也并不便宜。毕竟芯片、软件和生态的研发都是白花花的银子。

就像本文开头提到的微软给每台服务器部署 FPGA 一样，大规模量产的芯片价格就像沙子一样。到时候，能限制大模型推理算力的就只有能源了，就像区块链挖矿和通用 CPU 的云计算一样，都在找最便宜的电力供应。我在之前的一个采访中就表示，长期来看，能源和材料可能是制约大模型发展的关键。让我们期待廉价的大模型走进千家万户，真正改变人们的生活。

相关文章
大模型百川2技术报告细节分享
大模型来自面试的一些体会和分享
判断场景是否适合大模型
大模型微调技术报告汇总
大模型的幻觉问题
从零训练大模型教程
领域/场景大模型也太难训了吧
大模型面试八股含答案
大模型开源社区的原子弹Llama2
大模型训练的一些坑点和判断
大模型RLHF的trick
大模型评测，也太难了吧
大模型面试八股
大模型微调样本构造的trick
大模型训练太难了！

其他精彩文章翻阅公众号历史文章
包包算法笔记是包大人在班车通勤上，进行知识，职业，经验分享的地方。最白的话讲专业的知识。
回复“刷题”获取高效刷题经验，回复“面试”获取算法校招面试宝典。

想进讨论群的同学，加微信号logits，备注进群

点击原文跳转

阅读原文

喜欢此内容的人还喜欢

大模型来自面试的一些体会和分享

包包算法笔记

再现高配低价？13代酷睿+1TB+2.8K屏仅3799，堪称高性价比强者！

科技阿FENG

计算机专业买什么笔记本电脑好

笔记本电脑推荐选购