大模型 LLMs面试宝典

目录

一、LLMs Tokenizer 篇

Byte-Pair Encoding(BPE)篇

1 Byte-Pair Encoding(BPE) 如何构建词典?

1 准备足够的训练语料;以及期望的词表大小;

2 将单词拆分为字符粒度(字粒度),并在末尾添加后缀“”,统计单词频率

3 合并方式:统计每一个连续/相邻字节对的出现频率,将最高频的连续字节对合并为新字词;

4 重复第 3 步,直到词表达到设定的词表大小;或下一个最高频字节对出现频率为 1。 注:GPT2 、BART 和 LLaMA 就采用了BPE。

WordPiece 篇

WordPiece 与 BPE 异同点是什么?

本质上还是BPE 的思想。与BPE 最大区别在于:如何选择两个子词进行合并 BPE 是选择频次最大的相邻子词合并;

WordPiece 算法选择能够提升语言模型概率最大的相邻子词进行合并,来加入词表; 注:BERT 采用了WordPiece。

SentencePiece 篇

简单介绍一下 SentencePiece 思路?

把空格也当作一种特殊字符来处理,再用BPE 或者来构造词汇表。注:ChatGLM 、BLOOM 、PaLM 采用了SentencePiece。

对比篇

1 举例介绍一下不同大模型LLMs 的分词方式?

模型	词表大小	分词结果	长度
LLaMA	32000	['男', '<0xE5>', '<0x84>', '<0xBF>', '何', '不', '<0xE5>', '<0xB8>', '<0xA6>', '<0xE5>', '<0x90>', '<0xB4>', '<0xE9>', '<0x92>', '<0xA9>', '，', '收', '取', '关', '山', '五', '十', '州', '。']	24
Chinese LLaMA	49953	['男', '儿', '何', '不', '带', '吴', '钩', '，', '收取', '关', '山', '五十', '州', '。']	14
ChatGLM-6B	130528	['男儿', '何不', '带', '吴', '钩', ',', '收取', '关山', '五十', '州', '。']	11
ChatGLM2-6B	65024	['男', '儿', '何', '不', '带', '吴', '钩', '，', '收取', '关', '山', '五十', '州', '。']	14
Bloom	250880	['男', '儿', '何不', '带', '吴', '钩', '，', '收取', '关', '山', '五十', '州', '。']	13
Falcon	65024	['男', '儿', '', '', '', '不', '带', '吴', '', '', '', '', '', '收', '取', '', '', '山', '五', '十', '州', '。']	22

2 介绍一下不同大模型LLMs 的分词方式的区别?

模型	词表大小	中文平均token数	英文平均token数	中文处理时间(s)	英文处理时间(s)
LLaMA	32000	1.45	0.25	12.6	19.4
Falcon	65024	1.18	0.235	21.395	24.73
Chinese LLaMA	49953	0.62	0.249	8.65	19.12
ChatGLM-6B	130528	0.55	0.19	15.91	20.84
ChatGLM2-6B	65024	0.58	0.23	8.899	18.63
Bloom	250880	0.53	0.22	9.87	15.6

1. LLaMA 的词表是最小的，LLaMA 在中英文上的平均 token 数都是最多的，这意味着 LLaMA 对中英文分词都会比较碎，比较细粒度。尤其在中文上平均token 数高达 1.45， 这意味着LLaMA 大概率会将中文字符切分为2 个以上的token。
2. Chinese LLaMA 扩展词表后，中文平均token 数显著降低，会将一个汉字或两个汉字切 分为一个token，提高了中文编码效率。
3. ChatGLM-6B 是平衡中英文分词效果最好的tokenizer。由于词表比较大，中文处理时间 也有增加
4. BLOOM 虽然是词表最大的，但由于是多语种的，在中英文上分词效率与 ChatGLM-6B 基本相当。

二、大模型（LLMs）分布式训练面

1. 理论篇

1.1 想要训练1个LLM，如果只想用1张显卡，那么对显卡的要求是什 么?

显卡显存足够大， nB模型微调一般最好准备**20nGB以上的显存**。

1.2 如果有N张显存足够大的显卡，怎么加速训练?

数据并行（DP），充分利用多张显卡的算力。

1.3 如果显卡的显存不够装下一个完整的模型呢?

最直观想法，需要**分层加载，把不同的层加载到不同的GPU上**（accelerate的device_map） 也

就是常见的PP，流水线并行。

1.4 PP推理时，是一个串行的过程， 1个GPU计算，其他空闲，有没 有其他方式?

1. **横向切分：流水线并行（PP）**，也就是分层加载到不同的显卡上。
2. **纵向切分：张量并行（TP）**，在 DeepSpeed世界里叫模型并行（MP）

1.5 3种并行方式可以叠加吗?

是可以的， **DP+TP+PP**，这就是3D并行。**如果真有1个超大模型需要预训练， 3D并行那是必不可少的。** 毕竟显卡进化的比较慢，最大显存的也就是A100 80g。

单卡80g，可以完整加载小于40B的模型，但是训练时+梯度+优化器状态， 5B模型就是上限了，更别说activation的参数也要占显存， batch size还得大。而现在100亿以下（10B以下）的LLM只能叫small LLM。

1.6 Colossal-AI 有1D/2D/2.5D/3D，是什么情况?

Colossal-AI 的nD是针对张量并行，指的是TP的切分，对于矩阵各种切，和3D并行不是一回事。

1.7 除了3D并行有没有其他方式大规模训练?

可以使用更优化的数据并行算法FSDP（类似ZeRO3）或者直接使用 DeepSpeed ZeRO 。

1.8 有了ZeRO系列，为什么还需要3D并行?

根据ZeRO论文，尽管张量并行的显存更省一点，张量并行的通信量实在太高，只能限于节点内（有NVLINK）。如果节点间张量并行，显卡的利用率会低到5%

但是，根据Megatron-LM2的论文，当显卡数量增加到千量级， ZeRO3是明显不如3D并行的。

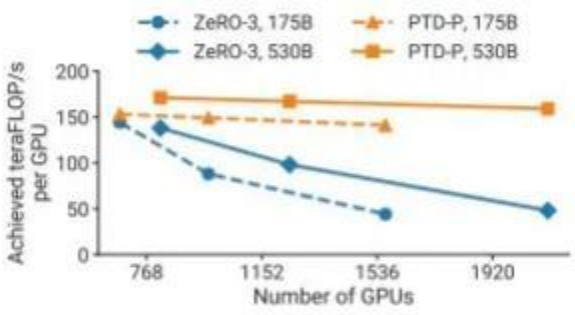


Figure 10: Throughput per GPU of PTD-P and ZeRO-3 for two different GPT models (the 175B GPT-3 model is shown with dotted lines, and the 530B model is shown with solid lines). Global batch sizes are fixed and ZeRO-3 is used without any model parallelism.

1.9 平民适不适合玩3D并行?

不适合。

3D并行的基础是，节点内显卡间NVLINK超高速连接才能上TP。有没有NVLINK都是个问题。 而且，节点间特殊的网络通常有400Gb/s？远超普通IDC内的万兆网络10Gb/s。

1.10 平民适不适合直接上多机多卡的ZeRO3（万兆网）？

不适合。

想象一下，当65B模型用Zero3，每一个step的每一张卡上需要的通信量是195GB（3倍参数量），也就 是1560Gb。万兆网下每步也要156s的通信时间，这画面太美。

2. 实践篇

2.1 假如有超多的8卡A100节点（DGX A100），如何应用3D并行策 略?

1. 首先，**张量并行**。3种并行方式里，张量并行（TP）对于GPU之间的通信要求最高，而节点内有 NVLINK 通信速度可以达到600GB/s。
2. 其次，**流水线并行**，每个节点负责一部分层，每35个节点组成一路完整的流水线，也就是一个完整 的模型副本，这里一个模型副本需280卡

3. 最后，**数据并行**，官方也做了8路， 10路， 12路的并行实验，分别使用280个节点， 350个节点和 420个节点。

参考 Megatron-Turing NLG 530B

集群规模越大，单个GPU利用率越低。

2.2 如果想构这样一个大规模并行训练系统，训练框架如何选?

可以参考Megatron-Turing NLG 530B ， NVIDIA Megatron-LM + Microsoft DeepSpeed

BLOOM **则是PP+DP用DeepSpeed，TP用Megatron-LM** 当然还有

一些其他的训练框架，在超大规模下或许也能work。

2.3 训练框架如何选?

下面这个图是bloom的一个实验， DP/TP/PP都能降显存，核心是要降到单卡峰值80g以下。 真

大模型就是要TP=8，充分利用NVLINK，然后优先PP，最后DP。

GPUs	Size	DP	TP	PP	MBS	Mem	TFLOPs	Notes
8	20B	1	8	1	1	68GB	107.48	02-17
80	200B	1	8	10	1	75GB	97.82	02-17
160	200B	2	8	10	1	53GB	96.19	02-17

然而假大模型（7B）比如LLaMA-7B，可以不用3D并行，直接用DeepSpeed ZeRO更方便，参考open-llama项目。

3. 并行化策略选择篇

3.1 单GPU

1. 显存够用： 直接用
2. 显存不够： 上offload，用cpu

3.2 单节点多卡

1. 显存够用（模型能装进单卡）： DDP或ZeRO
2. 显存不够： TP或者ZeRO或者PP

重点：没有NVLINK或者NVSwitch，也就是穷人模式，要用PP## 3.3 多节点多卡

如果节点间通信速度快（穷人的万兆网肯定不算）

ZeRO或者3D并行，其中3D并行通信量少但是对模型改动大。 如果节点间

通信慢，但显存又少。

DP+PP+TP+ZeRO-1

4. 问题篇

4.1 推理速度验证

ChatGML在V100单卡的推理耗时大约高出A800单卡推理的40%。

ChatGML推理耗时和问题输出答案的字数关系比较大，答案字数500字以内，A800上大概是每100字， 耗时1秒，V100上大概是每100字，耗时1.4秒。

1. ChatGML在A800单卡推理耗时统计

问题	运行次数	平均答案长度	平均耗时
给我介绍一下苹果公司,50个字	5	146.6	1.9458990097045898
给我介绍一下微软公司,50个字	5	104.6	1.2978283882141113
给我介绍一下苹果公司,100个字	5	165.4	2.0990972995758055
给我介绍一下微软公司,100个字	5	154.4	1.9092102527618409
给我介绍一下苹果公司,200个字	5	168.4	2.1302066326141356
给我介绍一下微软公司,200个字	5	208.8	2.6089860916137697
给我介绍一下苹果公司,300个字	5	443.4	5.4034130573272705
给我介绍一下微软公司,300个字	5	484.2	5.980589342
给我介绍一下苹果公司,500个字	5	525.4	6.246328496932984
给我介绍一下微软公司,500个字	5	591.6	6.961390399932862

1. ChatGML在V100单卡推理耗时统计

问题	运行次数	平均答案长度	平均耗时
给我介绍一下苹果公司,50个字	5	146.6	1.9458990097045898
给我介绍一下微软公司,50个字	5	104.6	1.2978283882141113
给我介绍一下苹果公司,100个字	5	165.4	2.0990972995758055
给我介绍一下微软公司,100个字	5	154.4	1.9092102527618409
给我介绍一下苹果公司,200个字	5	168.4	2.1302066326141356
给我介绍一下微软公司,200个字	5	208.8	2.6089860916137697
给我介绍一下苹果公司,300个字	5	443.4	5.4034130573272705
给我介绍一下微软公司,300个字	5	484.2	5.980589342
给我介绍一下苹果公司,500个字	5	525.4	6.246328496932984
给我介绍一下微软公司,500个字	5	591.6	6.961390399932862

1. 结论:
2. 训练效率方面: 多机多卡训练，增加训练机器可以线性缩短训练时间。
3. 推理性能方面:
4. ChatGML在V100单卡的推理耗时大约高出A800单卡推理的40%。
5. ChatGML推理耗时和问题输出答案的字数关系比较大，答案字数500字以内， A800上大概是每100 字，耗时1秒，V100上大概是每100字，耗时1.4秒。

4.2 并行化训练加速

可采用deepspeed进行训练加速，目前行业开源的大模型很多都是采用的基于deepspeed框架加速来进行模型训练的。如何进行deepspeed训练，可以参考基于deepspeed构建大模型分布式训练平台。

deepspeed在深度学习模型软件体系架构中所处的位置：

DL model—>train opitimization(deepspeed)—>train framework —> train instruction (cloud)— >GPU device

当然需要对比验证deepspeed的不同参数，选择合适的参数。分别对比stage 2,3进行验证，在GPU显 存够的情况下，最终使用stage 2。

4.3 deepspeed 训练过程，报找不主机

解决方法： deepspeed的关联的多机的配置文件， Hostfile 配置中使用ip，不使用hostname

4.4 为什么 多机训练效率不如单机?

多机训练可以跑起来，但是在多机上模型训练的速度比单机上还慢。

通过查看服务器相关监控，发现是网络带宽打满，上不去了，其他系统监控基本正常。原理初始的多机 之间的网络带宽是64Gps，后面把多机之间的网络带宽调整为800Gps，问题解决。

实验验证，多机训练的效率，和使用的机器数成线性关系，每台机器的配置一样，如一台GPU机器跑一 个epoch需要2小时，4台GPU机器跑一个epoch需要半小时。除了训练速度符合需求，多机训练模型的 loss下降趋势和单机模型训练的趋势基本一致，也符合预期。

4.5 多机训练不通， DeepSPeed配置问题

多机间NCCL 不能打通

1. 解决方法:

新建 .deepspeed_env 文件，写入如下内容

NCCL_IB_DISABLE=1

NCCL_ DEBUG= INFO

NCCL_SOCKET_IFNAME= eth0

NCCL_P2P_DISABLE=1

三、【大模型 LLMs 基础面 Plus】

Layer normalization-方法篇

1、Layer Norm 篇

1.1 Layer Norm 的计算公式写一下?

$$\mu = E(X) \leftarrow \frac{1}{H}\sum_{i=1}^{H} x_i$$

$$\sigma \leftarrow Var(x) = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2 + \epsilon}$$

$$y = \frac{x - E(x)}{\sqrt{Var(X) + \epsilon}} \cdot \gamma + \beta$$

gamma: 可训练的再缩放参数
beta: 可训练的再偏移参数

2、RMS Norm 篇 (均方根 Norm)

2.1 RMS Norm 的计算公式写一下?

$$RMS(x) = \sqrt{\frac{1}{H}\sum_{i=1}^{H} x_i^2}$$

$$x = \frac{x}{RMS(x)} \cdot \gamma$$

2.2 RMS Norm 相比于 Layer Norm 有什么特点?

RMS Norm 简化了 Layer Norm ，去除掉计算均值进行平移的部分。
对比LN，RMS Norm 的计算速度更快。效果基本相当，甚至略有提升。

3、Deep Norm 篇

3.1 Deep Norm 思路?

Deep Norm 方法在执行Layer Norm 之前，up-scale 了残差连接 (alpha>1)；另外，在初始化 阶段down-scale 了模型参数(beta<1)。

3.2 写一下 Deep Norm 代码实现?

```
def deepnorm(x):
    return LayerNorm(x * α + f(x))

def deepnorm_init(w):
    if w is ['ffn', 'v_proj', 'out_proj']:
        nn.init.xavier_normal_(w, gain=β)
    elif w is ['q_proj', 'k_proj']:
        nn.init.xavier_normal_(w, gain=1)
```

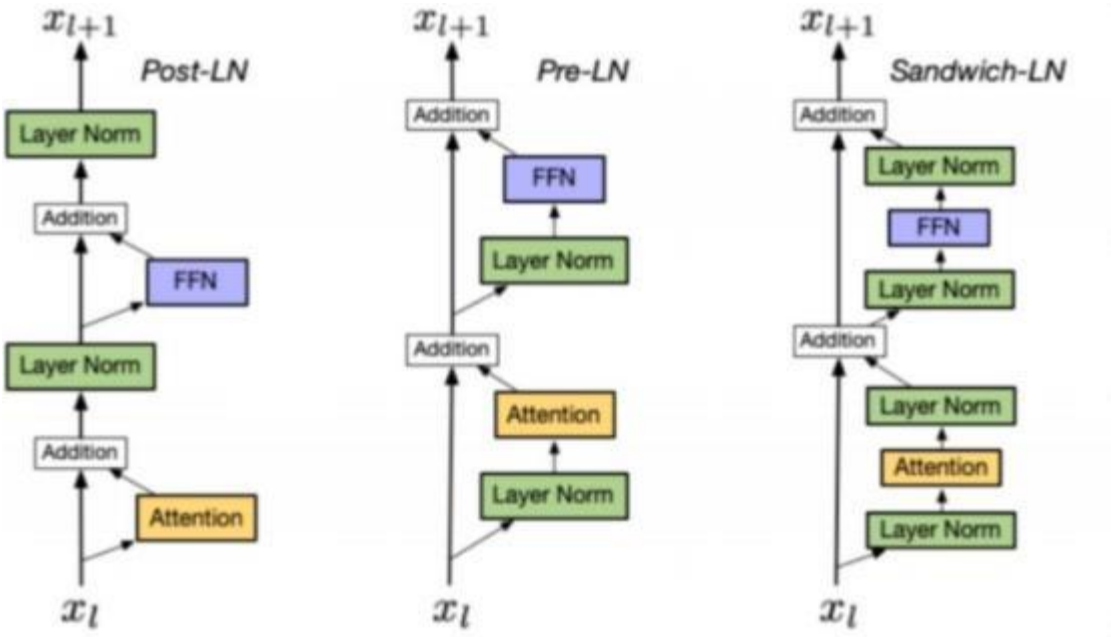
Deep Norm 有什么优点?

Deep Norm 可以缓解爆炸式模型更新的问题，把模型更新限制在常数，使得模型训练过程更 稳定。

Layer normalization-位置篇

LN 在 LLMs 中的不同位置有什么区别么？如有能介绍一下区别么?

回答：有，LN 在 LLMs 位置有以下几种：



Post LN：

位置：layer norm 在残差链接之后

缺点：PostLN 在深层的梯度范式逐渐增大，导致使用post-LN 的深层transformer 容易 出现训练不稳定的问题

Pre-LN：

位置：layer norm 在残差链接中

优点：相比于Post-LN，Pre LN 在深层的梯度范式近似相等，所以使用Pre-LN 的深层 transformer 训练更稳定，可以缓解训练不稳定问题

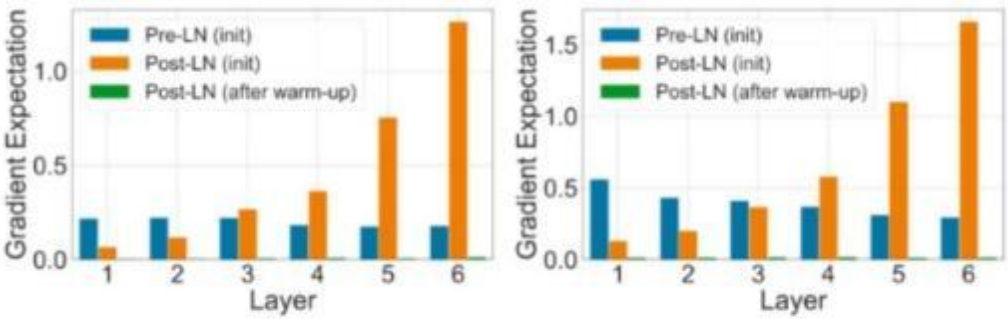
缺点：相比于 Post-LN，

Pre-LN 的模型效果略差 **Sandwich-LN：**

位置：在pre-LN 的基础上，额外插入了一个layer norm

优点：Cogview 用来避免值爆炸的问题

缺点：训练不稳定，可能会导致训练崩溃。



(a) W^1 in the FFN sub-layers (b) W^2 in the FFN sub-layers

Layer normalization 对比篇

LLMs 各模型分别用了哪种 Layer normalization?

模型	normalization
GPT3	Pre layer Norm
LLaMA	Pre RMS Norm
baichuan	Pre RMS Norm
ChatGLM-6B	Post Deep Norm
ChatGLM2-6B	Post RMS Norm
Bloom	Pre layer Norm
Falcon	Pre layer Norm

BLOOM 在 embedding 层后添加 layer normalization，有利于提升训练稳定性:但可能会带来 很大的性能损失。

LLMs 激活函数篇

1. 介绍一下FFN 块 计算公式?

$$FFN(x) = f(xW_1 + b_1)W_2 + b_2$$

2. 介绍一下 GeLU 计算公式?

$$GeLU(x) \approx 0.5x(1 + tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$$

3. 介绍一下 Swish 计算公式

$$Swish_\beta(x) = x \cdot \sigma(\beta x)$$

注：2 个可训练权重矩阵，中间维度为 4h

4. 介绍一下使用 GLU 线性门控单元的 FFN 块 计算公式?

$$GLU(x) = \sigma(xW + b) \otimes xV$$
$$FFN_{GLU} = (f(xW_1) \otimes xV)W_2$$

5. 介绍一下使用 GeLU 的 GLU 块 计算公式?

$$GeGLU(x) = GeLU(xW) \otimes xV$$

6. 介绍一下 使用 Swish 的 GLU 块 计算公式?

swuiGLU=swuishg(zrw)grv

注：3 个可训练权重矩阵，中间维度为 4h*2/3

各LLMs 都使用哪种激活函数?

模型	激活函数
GPT3	GeLU
LLaMA	SwiGLU
LLaMA2	SwiGLU
baichuan	SwiGLU
ChatGLM-6B	GeLU
ChatGLM2-6B	SwiGLU
Bloom	GeLU
Falcon	GeLU

LLMs 注意力机制 优化篇

1 传统 Attention 存在哪些问题?

传统 Attention 存在上下文长度约束问题；传统 Attention 速度慢，内存占用大；

2 Attention 优化方向?

提升上下文长度

加速、减少内存占用

3 Attention 变体有哪些?

稀疏 attention。将稀疏偏差引入 attention 机制可以降低了复杂性；

线性化attention。解开 attention 矩阵与内核特征图，然后以相反的顺序计算 attention 以实现线性复杂度；

原型和内存压缩。这类方法减少了查询或键值记忆对的数量，以减少注意力矩阵的大小； 低阶 self-Attention。这一系列工作捕获了 self-Attention 的低阶属性；

Attention 与先验。该研究探索了用先验 attention 分布来补充或替代标准 attention； **改进多头机制**。该系列研究探索了不同的替代多头机制。

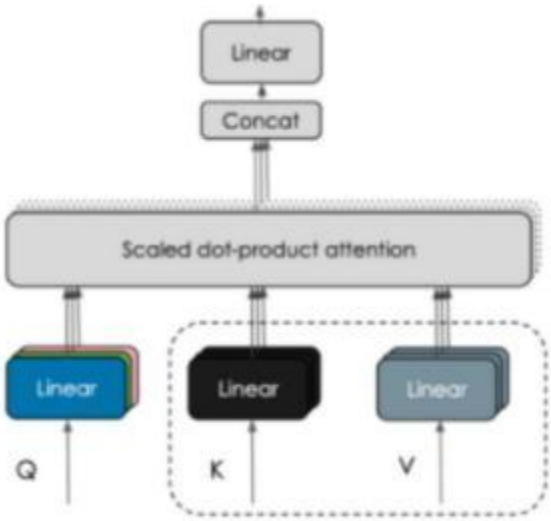
4 Multi-Query Attention 篇

4.1 Multi-head Attention 存在什么问题?

训练过程：不会显著影响训练过程，训练速度不变，会引起非常细微的模型效果损失；
推理过程：反复加载巨大的 KV cache， 导致内存开销大，性能是内存受限；

4.2 介绍一下 Multi-Query Attention?

Multi-Query Attention 在所有注意力头上共享 key 和 value.



4.3 对比一下 Multi-head Attention 和 Multi-Query Attention?

Multi-head Attention：每个注意力头都有各自的query、key 和value。 Multi-query Attention: 在所有的注意力头上共享key 和value。

模型	n_heads	head_dim	FFN中间维度	维度h
LLaMA	32	128	11008	4096
baichuan	32	128	11008	4096
ChatGLM-6B	32	128	4h, 16384	4096
ChatGLM2-6B	32	128	13696	4096
Bloom	32	128	4h, 16384	4096
Falcon	71	64	4h, 18176	4544

Falcon、PaLM、ChatGLM2-6B 都使用了Multi-query Attention，但有细微差别。 为了保持参数量一致，

Falcon: 把隐藏维度从4096 增大到了4544。多余的参数量分给了Attention 块和FFN 块 ChatGLM2: 把FFN 中间维度从 11008 增大到了 13696。多余的参数分给了FFN 块

4.4 Multi-Query Attention 这样做的好处是什么?

减少 KV cache 的大小，减少显存占用，提升推理速度。

4.5 有哪些模型是使用 Multi-Query Attention?

代表模型：PaLM 、ChatGLM2 、Falcon 等

5 Grouped-query Attention 篇

5.1 什么是 Grouped-query Attention?

Grouped query attention: 介于multi head 和multi query 之间，多个key 和value。

5.2 有哪些大模型使用 Grouped-query Attention?

ChatGLM2 ，LLaMA2-34B/70B 使用了Grouped query attention。

6 Flash Attention 篇

核心：用分块softmax 等价替代传统softmax

优点：节约HBM，高效利用SRAM，省显存，提速度

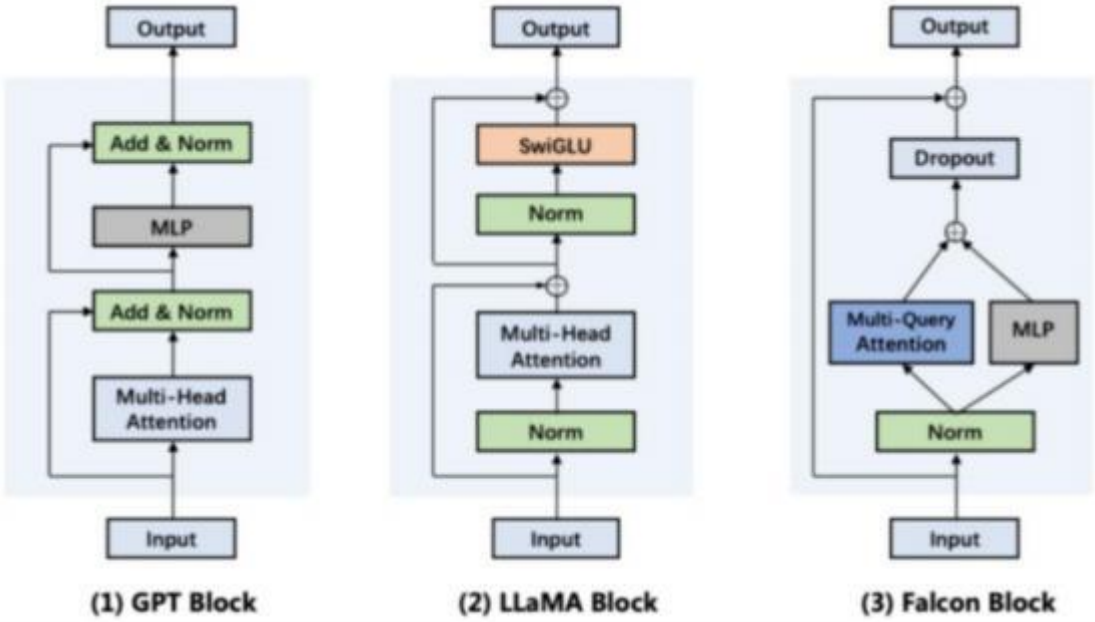
代表模型：Meta 推出的开源大模型LLaMA，阿联酋推出的开源大模型Falcon 都使用了Flash Attention 来加速计算和节省显存

关键词：HBM、SRAM、分块 Softmax、重计算、Kernel 融合。

7 并行 transformer block

用并行公式替换了串行，提升了 15%的训练速度。

在8B 参数量规模，会有轻微的模型效果损失;在62B 参数量规模，就不会损失模型效果。 Falcon 、PaLM 都使用了该技术来加速训练



LLMs 损失函数篇

1 介绍一下 KL 散度?

KL（Kullback-Leibler）散度衡量了两个概率分布之间的差异。其公式为：

$$D_{KL}(P//Q) = -\sum_{x \in X} P(x) \log \frac{1}{P(x)} + \sum_{x \in X} P(x) \log \frac{1}{Q(x)}$$

2 交叉熵损失函数写一下，物理意义是什么?

交叉熵损失函数（Cross-Entropy Loss Function）是用于度量两个概率分布之间的差异的一种 损失函数。在分类问题中，它通常用于衡量模型的预测分布与实际标签分布之间的差异。

$$H(p,q) = -\sum_{i=1}^{N} p_i \log(q_i) - (1-p_i) \log(1-q_i)$$

3 KL 散度与交叉熵的区别?

KL 散度指的是相对熵，KL 散度是两个概率分布 P 和 Q 差别的非对称性的度量。KL 散度 越小表示两个分布越接近。也就是说 KL 散度是不对称的，且 KL 散度的值是非负数。（也就是熵和交叉熵的差）

- 交叉熵损失函数是二分类问题中最常用的损失函数，由于其定义出于信息学的角 度，可以泛化到多分类问题中。
- KL 散度是一种用于衡量两个分布之间差异的指标，交叉熵损失函数是KL 散度的 一种特殊形式。在二分类问题中，交叉熵函数只有一项，而在多分类问题中有多项。

I

4 多任务学习各loss 差异过大怎样处理?

多任务学习中，如果各任务的损失差异过大，可以通过动态调整损失权重、使用任务特定的 损失函数、改变模型架构或引入正则化等方法来处理。目标是平衡各任务的贡献，以便更好 地训练模型。

5 分类问题为什么用交叉熵损失函数不用均方误差（MSE）?

交叉熵损失函数通常在分类问题中使用，而均方误差（MSE）损失函数通常用于回归问题。 这是因为分类问题和回归问题具有不同的特点和需求。

分类问题的目标是将输入样本分到不同的类别中，输出为类别的概率分布。交叉熵损失函数 可以度量两个概率分布之间的差异，使得模型更好地拟合真实的类别分布。它对概率的细微 差异更敏感，可以更好地区分不同的类别。此外，交叉熵损失函数在梯度计算时具有较好的 数学性质，有助于更稳定地进行模型优化。

相比之下，均方误差（MSE）损失函数更适用于回归问题，其中目标是预测连续数值而不是 类别。MSE 损失函数度量预测值与真实值之间的差异的平方，适用于连续数值的回归问题。 在分类问题中使用MSE 损失函数可能不太合适，因为它对概率的微小差异不够敏感，而且 在分类问题中通常需要使用激活函数（如 sigmoid 或 softmax）将输出映射到概率空间，使 得MSE 的数学性质不再适用。

综上所述，交叉熵损失函数更适合分类问题，而MSE 损失函数更适合回归问题。

6 什么是信息增益?

信息增益是在决策树算法中用于选择最佳特征的一种评价指标。在决策树的生成过程中，选 择最佳特征来进行节点的分裂是关键步骤之一，信息增益可以帮助确定最佳特征。

信息增益衡量了在特征已知的情况下，将样本集合划分成不同类别的纯度提升程度。它基于 信息论的概念，使用熵来度量样本集合的不确定性。具体而言，信息增益是原始集合的熵与 特定特征下的条件熵之间的差异。

在决策树的生成过程中，选择具有最大信息增益的特征作为当前节点的分裂标准，可以将样 本划分为更加纯净的子节点。

信息增益越大，意味着使用该特征进行划分可以更好地减少样 本集合的不确定性，提高分类的准确性

7 多分类的分类损失函数(Softmax)?

多分类的分类损失函数采用Softmax 交叉熵（Softmax Cross Entropy）损失函数。Softmax 函 数可以将输出值归一化为概率分布，用于多分类问题的输出层。Softmax 交叉熵损失函数可
以写成:

$$-\sum_{i=1}^{n} y_i \log(p_i)$$

注：其中，n 是类别数，yi 是第i 类的真实标签，pi 是第i 类的预测概率。

8 softmax 和交叉熵损失怎么计算，二值交叉熵呢?

softmax 计算公式如下:

$$y = \frac{e^{f_i}}{\sum_j e^{fj}}$$

多分类交叉熵:

$$L = \frac{1}{N}\sum_i L_i = -\frac{1}{N}\sum_i \sum_{c=1}^{M} y_{ic} \log$$

其中:

(p_{ic})

- M ——类别的数量
- y_{ic} 一符号函数（0 或 1），如果样本 i 的真实类别等于 c 取 1，否则取 0
- p_{ic} 一观测样本 i 属于类别 c 的预测概率

二分类交叉熵:

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i$$

$$-[y_i \cdot \log(p_i) + (1-y_i) \cdot \log(1-p_i)]$$

- y_i — 表示样本 i 的label, 正类为 1，负类为 0
- p_i — 表示样本 i 预测为正类的概率

9 如果softmax 的e 次方超过float 的值了怎么办?

将分子分母同时除以 x 中的最大值，可以解决。

$$\tilde{x}_k = \frac{e^{xk-\max(x)}}{e^{x_1-\max(x)}+e^{x_2-\max(x)}+\ldots+e^{x_k-\max(x)}+\ldots+e^{x_n-\max(x)}}$$

LLMs 相似度函数篇

1 除了cosin 还有哪些算相似度的方法

除了余弦相似度（cosine similarity）之外，常见的相似度计算方法还包括欧氏距离、曼哈顿 距离、Jaccard 相似度、皮尔逊相关系数等。

2 了解对比学习嘛?

对比学习是一种无监督学习方法，通过训练模型使得相同样本的表示更接近，不同样本的表 示更远离，从而学习到更好的表示。对比学习通常使用对比损失函数，例如 Siamese 网络、 Triplet 网络等，用于学习数据之间的相似性和差异性。

3 对比学习负样本是否重要? 负样本构造成本过高应该怎么解决?

对比学习中负样本的重要性取决于具体的任务和数据。负样本可以帮助模型学习到样本之间 的区分度，从而提高模型的性能和泛化能力。然而，负样本的构造成本可能会较高，特别是 在一些领域和任务中。

为了解决负样本构造成本过高的问题，可以考虑以下方法：

降低负样本的构造成本：通过设计更高效的负样本生成算法或采样策略，减少负样本的构造 成本。例如，可以利用数据增强技术生成合成的负样本，或者使用近似采样方法选择与正样 本相似但不相同的负样本。

确定关键负样本：根据具体任务的特点，可以重点关注一些关键的负样本，而不是对所有负 样本进行详细的构造。这样可以降低构造成本，同时仍然能够有效训练模型。

迁移学习和预训练模型：利用预训练模型或迁移学习的方法，可以在其他领域或任务中利用 已有的负样本构造成果，减少重复的负样本构造工作。

【大模型LLMs 训练经验面Plus】

LLMs 训练经验帖

1 分布式训练框架选择?

多用 DeepSpeed，少用 Pytorch 原生的 torchrun。在节点数量较少的情况下，使用何种训练 框架并不是特别重要；然而，一旦涉及到数百个节点，DeepSpeed 显现出其强大之处，其简 便的启动和便于性能分析的特点使其成为理想之选。

2 LLMs 训练时有哪些有用的建议?

l 弹性容错和自动重启机制

大模型训练不是以往那种单机训个几小时就结束的任务，往往需要训练好几周甚至好几个 月，这时候你就知道能稳定训练有多么重要。弹性容错能让你在机器故障的情况下依然继续 重启训练；自动重启能让你在训练中断之后立刻重启训练。毕竟，大模型时代，节约时间就 是节约钱。

l 定期保存模型

训练的时候每隔一段时间做个checkpointing，这样如果训练中断还能从上次的断点来恢复训 练。

l 想清楚再开始训练

训练一次大模型的成本很高的。在训练之前先想清楚这次训练的目的，记录训练参数和中间 过程结果，少做重复劳动。

l 关注GPU 使用效率

有时候，即使增加了多块 A100 GPU，大型模型的训练速度未必会加快，这很可能是因为 GPU 使用效率不高，尤其在多机训练情况下更为明显。仅仅依赖nvidia-smi 显示的GPU 利 用率并不足以准确反映实际情况，因为即使显示为 100%，实际 GPU 利用率也可能不是真 正的 100%。要更准确地评估 GPU 利用率，需要关注 TFLOPS 和吞吐率等指标，这些监控 在 DeepSpeed 框架中都得以整合。

l 不同的训练框架对同一个模型影响不同

对于同一模型，选择不同的训练框架，对于资源的消耗情况可能存在显著差异（比如使用 Huggingface Transformers 和DeepSpeed 训练OPT-30 相对于使用Alpa 对于资源的消耗会低 不少）。

l 环境问题

针对已有的环境进行分布式训练环境搭建时，一定要注意之前环境的python、pip、virtualenv、 setuptools 的版本。不然创建的虚拟环境即使指定对了Python 版本，也可能会遇到很多安装 依赖库的问题（GPU 服务器能够访问外网的情况下，建议使用Docker 相对来说更方便）。

l 升级GLIBC 等底层库问题

遇到需要升级GLIBC 等底层库需要升级的提示时，一定要慎重，不要轻易升级，否则，可 能会造成系统宕机或很多命令无法操作等情况。

3 模型大小如何选择?

进行大模型模型训练时，先使用小规模模型（如：OPT-125m/2.7b）进行尝试，然后再进行 大规模模型（如：OPT-13b/30b...)的尝试,便于出现问题时进行排查。目前来看,业界也是 基于相对较小规模参数的模型(6B/7B/13B）进行的优化，同时，13B 模型经过指令精调之 后的模型效果已经能够到达GPT4 的90%的效果。

4 加速卡如何选择?

于一些国产AI 加速卡，目前来说，坑还比较多，如果时间不是时间非常充裕，还是尽量选 择Nvidia 的AI 加速卡。

【大模型（LLMs）RAG 检索增强生成面】

RAG（Retrieval-Augmented Generation）面

1 RAG 基础面

1.1 为什么大模型需要外挂（向量） 知识库?

如何将外部知识注入大模型，最直接的方法：利用外部知识对大模型进行微调。

思路：构建几十万量级的数据，然后利用这些数据 对大模型进行微调，以将 额外知识注入 大模型

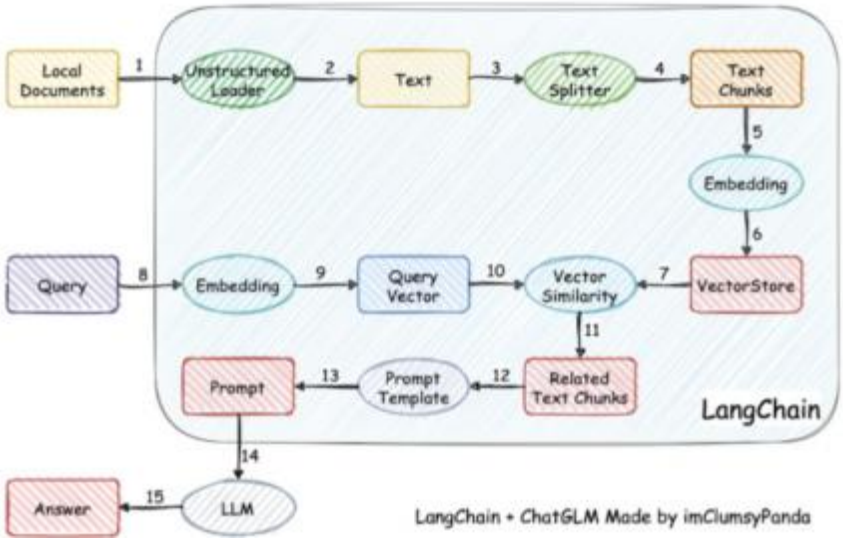
优点：简单粗暴 **缺点**:

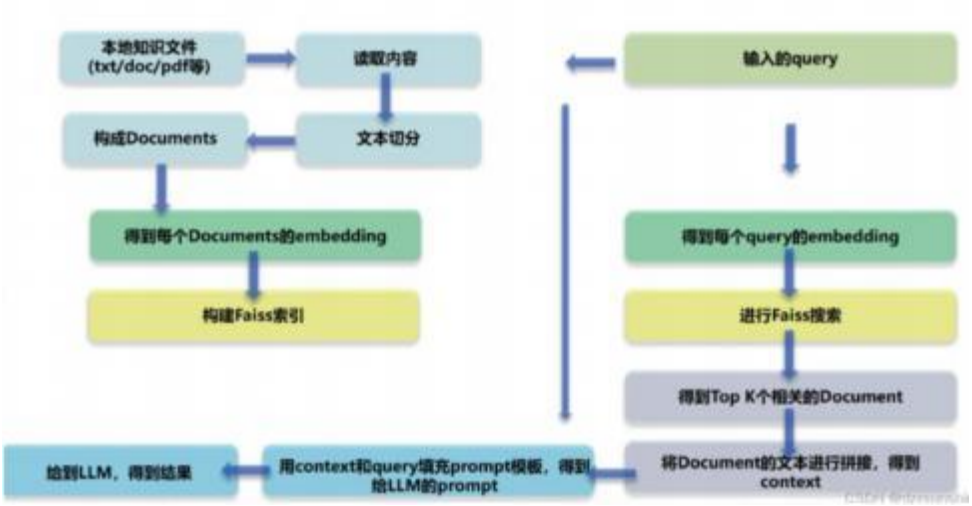
这几十万量级的数据 并不能很好的将额外知识注入大模型； 训练成本昂贵。不仅需要 多卡并行，还需要 训练很多天；

既然大模型微调不是将外部知识注入大模型的最优方案，那是否有其它可行方案?

1.2. RAG 思路是怎么样?

- 加载文件
- 读取文本
- 文本分割
- 文本向量化
- 问句向量化
- 在文本向量中匹配出与问句向量最相似的top k 个
- 匹配出的文本作为上下文和问题一起添加到 prompt 中
- 提交给 LLM 生成回答





1.3. RAG 核心技术是什么?

RAG 核心技术：embedding

思路：将用户知识库内容经过 embedding 存入向量知识库，然后用户每一次提问也会经过 embedding，利用向量相关性算法（例如余弦算法）找到最匹配的几个知识库片段，将这些 知识库片段作为上下文，与用户问题一起作为 promt 提交给 LLM 回答。

RAG prompt 模板如何构建?

已知信息: {context}

根据上述已知信息，简洁和专业的来回答用户的问题。如果无法从中得到答案，请说 “根 据已知信息无法回答该问题” 或 “没有提供足够的相关信息”，不允许在答案中添加编造 成分，答案请使用中文。
问题是：{question}

2 RAG 优化面

痛点 1：文档切分粒度不好把控，既担心噪声太多又担心语义信息丢失

问题描述
问题 1：如何让 LLM 简要、准确回答细粒度知识?

用户：2023 年我国上半年的国内生产总值是多少?
LLM：根据文档，2023 年的国民生产总值是 593034 亿元。
需求分析：一是简要，不要有其他废话。二是准确，而不是随意编造。 **问题2：如何让 LLM 回**

答出全面的粗粒度（跨段落）知识?

用户：根据文档内容，征信中心有几点声明?
LLM：根据文档内容，有三点声明，分别是：一、……；二……；三……。 **需求分析:**
要实现语义级别的分割，而不是简单基于html 或者pdf的换行符分割。

笔者发现目前的痛点是文档分割不够准确，导致模型有可能只回答了两点，而实际上是因为 向量相似度召回的结果是残缺的。

有人可能会问，那完全可以把切割粒度大一点，比如每 10 个段落一分。但这样显然不是最 优的，因为召回片段太大，噪声也就越多。LLM 本来就有幻觉问题，回答得不会很精准（笔 者实测也发现如此）。

所以说，我们的文档切片最好是按照语义切割。

解决方案：

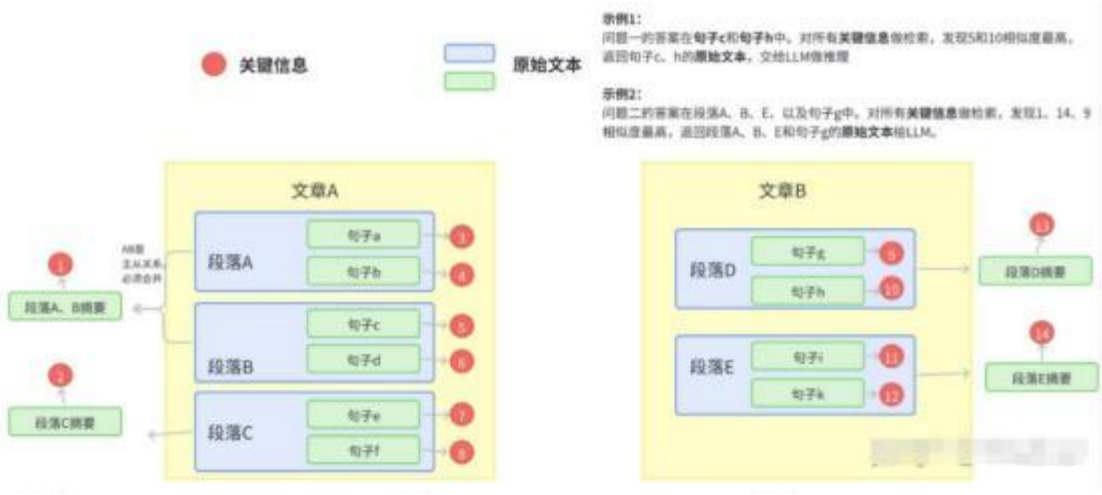
思想（原则）

基于LLM 的文档对话架构分为两部分，先检索，后推理。重心在检索（推荐系统），推理交 给LLM 整合即可。

而检索部分要满足三点 **①尽可能提高召回率，②尽可能减少无关信息；③速度快。**

将所有的文本组织成二级索引，第一级索引是 [关键信息]，第二级是 [原始文本]，二者一 一映射。

检索部分只对关键信息做 embedding，参与相似度计算，把召回结果映射的 原始文本交给 LLM。主要架构图如下：



如何构建关键信息?

首先从架构图可以看到，句子、段落、文章都要关键信息，如果为了效率考虑，可以不用对 句子构建关键信息。

1 文章的切分及关键信息抽取

关键信息： 为各语义段的关键信息集合，或者是各个子标题语义扩充之后的集合（pdf 多 级

四、大模型LLMs推理加速篇

1. 当前优化模型最主要技术手段有哪些?

1. 算法层面：蒸馏、量化
2. 软件层面：计算图优化、模型编译
3. 硬件层面： FP8（ NVIDIA H系列GPU开始支持FP8，兼有fp16的稳定性和int8的速度）

2. 推理加速框架有哪一些? 都有什么特点?

1. FasterTransformer：英伟达推出的FasterTransformer不修改模型架构而是在计算加速层面优化 Transformer 的 encoder 和 decoder 模块。具体包括如下：
2. 尽可能多地融合除了 GEMM 以外的操作
3. 支持 FP16、 INT8、 FP8
4. 移除 encoder 输入中无用的 padding 来减少计算开销
5. TurboTransformers：腾讯推出的 TurboTransformers 由 computation runtime 及 serving framework 组成。加速推理框架适用于 CPU 和 GPU，最重要的是，它可以无需预处理便可处理变 长的输入序列。具体包括如下：
6. 与 FasterTransformer 类似，它融合了除 GEMM 之外的操作以减少计算量
7. smart batching，对于一个 batch 内不同长度的序列，它也最小化了 zero-padding 开销
8. 对 LayerNorm 和 Softmax 进行批处理，使它们更适合并行计算
9. 引入了模型感知分配器，以确保在可变长度请求服务期间内存占用较小

3 vLLM 篇

3.1 vLLM 的 功能有哪些?

1. Continuous batching：有iteration-level的调度机制，每次迭代batch大小都有所变化，因此vLLM在大量查询下仍可以很好的工作；
2. PagedAttention：受操作系统中虚拟内存和分页的经典思想启发的注意力算法，这就是模型加速的 秘诀

3.2 vLLM 的 优点有哪些?

1. **文本生成的速度**：实验多次，发现vLLM的推理速度是最快的；
2. **高吞吐量服务**：支持各种解码算法，比如parallel sampling, beam search等；
3. **与OpenAI API兼容**：如果使用OpenAI API，只需要替换端点的URL即可；

3.3 vLLM 的 缺点有哪些?

1. **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；
2. **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。

3. **缺少权重量化**：有时， LLM可能不需要使用GPU内存，这对于减少GPU内存消耗至关重要。

3.4 vLLM 离线批量推理?

```
# pip install vllm

from vllm import LLM, SamplingParams

prompts = [

•     "Funniest joke ever:",

•     "The capital of France is",

•     "The future of AI is",

]

sampling_params = SamplingParams(temperature=0.95, top_p=0.95, max_tokens=200) llm =

LLM(model="huggyllama/llama-13b")

outputs = llm.generate(prompts, sampling_params)

for output in outputs:

•     prompt = output.prompt

•     generated_text = output.outputs[0].text

•     print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

3.5 vLLM API Server?

```
# Start the server:

python -m vllm.entrypoints.api_server --env MODEL_NAME=huggyllama/llama-13b

\# Query the model in shell:
```

```
curl http://localhost:8000/generate \
    -d '{
        "prompt": "Funniest joke ever:",
        "n": 1,
        "temperature": 0.95,
        "max_tokens": 200
    }'
```

4 Text generation inference 篇

4.1 介绍一下 Text generation inference?

Text generation inference是用于文本生成推断的Rust、 Python和gRPC服务器， 在HuggingFace中已 有LLM 推理API使用。

4.2 Text generation inference 的 功能有哪些?

1. 内置服务评估：可以监控服务器负载并深入了解其性能；
2. 使用flash attention（和v2）和Paged attention优化transformer推理代码： 并非所有模型都内置 了对这些优化的支持，该技术可以对未使用该技术的模型可以进行优化；

4.3 Text generation inference 的 优点有哪些?

1. 所有的依赖项都安装在Docker中：会得到一个现成的环境；
2. 支持HuggingFace模型：轻松运行自己的模型或使用任何HuggingFace模型中心；
3. 对模型推理的控制：该框架提供了一系列管理模型推理的选项，包括精度调整、量化、张量并行 性、重复惩罚等；

4.4 Text generation inference 的 缺点有哪些?

1. 缺乏对适配器的支持：需要注意的是，尽管可以使用适配器部署LLM（可以参考 https://www.youtube.com/watch?v=HI3cYN0c9ZU），但目前还没有官方支持或文档；
2. 从源代码（Rust+CUDA内核）编译：对于不熟悉Rust的人，将客户化代码纳入库中变得很有挑战 性；
3. 文档不完整：所有信息都可以在项目的自述文件中找到。尽管它涵盖了基础知识，但必须在问题或 源代码中搜索更多细节；

4.5 Text generation inference 的 使用docker运行web server?

```
mkdir data

docker run --gpus all --shm-size 1g -p 8080:80 \

-v data:/data ghcr.io/huggingface/text-generation-inference:0.9 \ --model-id

   huggyllama/llama-13b \

   --num-shard 1
```

标题识别及提取见下一篇文章）

语义切分方法 1：利用NLP 的篇章分析（discourse parsing）工具，提取出段落之间的主要关 系，譬如上述极端情况 2 展示的段落之间就有从属关系。把所有包含主从关系的段落合并成 一段。 这样对文章切分完之后保证每一段在说同一件事情.

语义切分方法 2：除了discourse parsing 的工具外，还可以写一个简单算法**利用BERT 等模 型来实现语义分割。**BERT 等模型在预训练的时候采用了NSP（next sentence prediction）的 训练任务，因此 BERT 完全可以判断两个句子（段落）是否具有语义衔接关系。这里我们可 以设置相似度阈值 t，从前往后依次判断相邻两个段落的相似度分数是否大于t，如果大于则 合并，否则断开。当然算法为了效率，可以采用二分法并行判定，模型也不用很大，笔者用 BERT-base-Chinese 在中文场景中就取得了不错的效果。

2 语义段的切分及段落（句子）关键信息抽取

如果向量检索效率很高，获取语义段之后完全可以按照真实段落及句号切分，以缓解细 粒度知识点检索时大语块噪声多的场景。当然，关键信息抽取笔者还有其他思路。

方法 1：利用 NLP 中的成分句法分析（constituency parsing）工具和命名实体识别（NER） 工具提取

成分句法分析（constituency parsing）工具：可以提取核心部分（名词短语、动词短语……）；命名实体识别（NER）工具：可以提取重要实体（货币名、人名、企业名……）。

譬如说:

原始文本：MM 团队的成员都是精英，核心成员是前谷歌高级产品经理张三，前 meta 首席技术官李四……

关键信息：（MM 团队，核心成员，张三，李四）

方法2：可以用语义角色标注（Semantic Role Labeling）来分析句子的谓词论元结构，提取 “谁对谁做了什么”的信息作为关键信息。

方法3：直接法。其实NLP 的研究中本来就有关键词提取工作（Keyphrase Extraction）。也 有一个成熟工具可以使用。一个工具是 HanLP ，中文效果好，但是付费，免费版调用次数 有限。还有一个开源工具是KeyBERT，英文效果好，但是中文效果差。

方法4：垂直领域建议的方法。以上两个方法在垂直领域都有准确度低的缺陷，垂直领域可 以仿照ChatLaw 的做法，即：训练一个生成关键词的模型。ChatLaw 就是训练了一个KeyLLM。

常见问题

句子、语义段、之间召回不会有包含关系吗，是否会造成冗余?

回答：会造成冗余，但是笔者试验之后回答效果很好，无论是细粒度知识还是粗粒度（跨段 落）知识准确度都比Longchain 粗分效果好很多，对这个问题笔者认为可以优化但没必要

痛点2：在基于垂直领域表现不佳

模型微调：一个是对embedding 模型的基于垂直领域的数据进行微调；一个是对LLM 模型 的基于垂直领域的数据进行微调；

痛点 3：langchain 内置问答分句效果不佳问题

文档加工:

一种是使用更好的文档拆分的方式（如项目中已经集成的达摩院的语义识别的模型及进 行拆分）；

一种是改进填充的方式，判断中心句上下文的句子是否和中心句相关，仅添加相关度高 的句子；

另一种是文本分段后，对每段分别及进行总结，基于总结内容语义及进行匹配；

痛点4：如何 尽可能召回与query 相关的Document 问题

问题描述：如何通过得到query 相关性高的context，即与 query 相关的Document 尽可能多 的能被召回；

解决方法:

将本地知识切分成Document 的时候，需要考虑Document 的长度、Document embedding 质 量和被召回Document 数量这三者之间的相互影响。在文本切分算法还没那么智能的情况下， 本地知识的内容最好是已经结构化比较好了，各个段落之间语义关联没那么强。Document 较短的情况下，得到的 Document embedding 的质量可能会高一些，通过 Faiss 得到的 Document 与query 相关度会高一些。

使用Faiss 做搜索，前提条件是有高质量的文本向量化工具。因此最好是能基于本地知识对 文本向量化工具进行Finetune。另外也可以考虑将ES 搜索结果与Faiss 结果相结合。

痛点5：如何让LLM 基于query 和context 得到高质量的response

问题描述：如何让LLM 基于query 和context 得到高质量的response **解决方法:**

尝试多个的prompt 模版，选择一个合适的，但是这个可能有点玄学 用与本地知识问答相关的语料，对 LLM 进行Finetune。

痛点6：embedding 模型在表示text chunks 时偏差太大问题

问题描述:

一些开源的 embedding 模型本身效果一般，尤其是当text chunk 很大的时候，强行变成一个 简单的vector 是很难准确表示的，开源的模型在效果上确实不如openai Embeddings；

多语言问题，paper 的内容是英文的，用户的 query 和生成的内容都是中文的，这里有个语 言之间的对齐问题，尤其是可以用中文的 query embedding 来从英文的 text chunking embedding 中找到更加相似的top-k 是个具有挑战的问题

解决方法:

用更小的text chunk 配合更大的topk 来提升表现，毕竟smaller text chunk 用embedding 表示 起来noise 更小，更大的topk 可以组合更丰富的context 来生成质量更高的回答；

多语言的问题，可以找一些更加适合多语言的 embedding 模型

痛点7：不同的 prompt，可能产生完全不同的效果问题

问题描述：prompt 是个神奇的东西，不同的提法，可能产生完全不同的效果。尤其是指令， 指令型 llm 在训练或者微调的时候，基本上都有个输出模板，这个如果前期没有给出 instruction data 说明，需要做很多的尝试，尤其是你希望生成的结果是按照一定格式给出的， 需要做更多的尝试

痛点8： llm 生成效果问题

问题描述：LLM 本质上是个“接茬”机器，你给上句，他补充下一句。但各家的LLM 在理 解 context 和接茬这两个环节上相差还是挺多的。最早的时候，是用一个付费的GPT 代理作 为LLM 来生成内容，包括解读信息、中文标题和关键词，整体上来看可读性会强很多，也 可以完全按照给定的格式要求生成相应的内容，后期非常省心；后来入手了一台mac m2， 用llama.cpp 在本地提供LLM 服务，模型尝试了chinese-llama2-alpaca 和baichuan2，量化用 了 Q6_K level，据说性能和 fp16 几乎一样，作为开源模型，两个表现都还可以。前者是在 llama2 的基础上，用大量的中文数据进行了增量训练，然后再用alpaca 做了指令微调，后者 是开源届的当红炸子鸡。但从 context 的理解上，两者都比较难像GPT 那样可以完全准确地 生成我希望的格式，baichuan2 稍微好一些。我感觉，应该是指令微调里自带了一些格式， 所以生成的时候有点“轴”。

解决思路：可以选择一些好玩的开源模型，比如 llama2 和 baichuan2，然后自己构造一些 domain dataset，做一些微调的工作，让llm 更听你的话

痛点9：如何更高质量地召回context 喂给llm

问题描述：初期直接调包langchain 的时候没有注意，生成的结果总是很差，问了很多Q 给 出的A 乱七八糟的，后来一查，发现召回的内容根本和Q 没啥关系

解决思路：更加细颗粒度地来做 recall，当然如果是希望在学术内容上来提升质量，学术相 关的embedding 模型、指令数据，以及更加细致和更具针对性的pdf 解析都是必要的。

参考：PDFTriage: Question Answering over Long, Structured Documents

3 RAG 评测面

3.1 为什么需要对 RAG 进行评测?

在探索和优化 RAG（检索增强生成器）的过程中，如何有效评估其性能已经成为关键问题。

3.2 RAG 有哪些评估方法?

主要有两种方法来评估 RAG 的有效性：独立评估和端到端评估。

独立评估

介绍：独立评估涉及对检索模块和生成模块（即阅读和合成信息）的评估。

(1) 检索模块：

介绍：评估 RAG 检索模块的性能通常使用一系列指标，这些指标用于衡量系统（如搜 索引擎、推荐系统或信息检索系统）在根据查询或任务排名项目的有效性。

指标：命中率（Hit Rate)、平均排名倒数（MRR)、归一化折扣累积增益（NDCG)、**精 确度（Precision) ** 等。

(2) 生成模块：

介绍：生成模块指的是将检索到的文档与查询相结合，形成增强或合成的输入。这与最 终答案或响应的生成不同，后者通常采用端到端的评估方式。

评估指标：关注上下文相关性，即检索到的文档与查询问题的关联度。

端到端评估

介绍：对 RAG 模型对特定输入生成的最终响应进行评估，涉及模型生成的答案与输入 查询的相关性和一致性。

无标签的内容评估：

评价指标：答案的准确性、相关性和无害性

有标签的内容评估：

评价指标：准确率（Accuracy） 和精确匹配（EM）

3 RAG 有哪些关键指标和能力？

评估 RAG 在不同下游任务和不同检索器中的应用可能会得到不同的结果。然而，一些学术 和工程实践已经开始关注 RAG 的通用评估指标和有效运用所需的能力。

关键指标：集中于三个关键指标：答案的准确性、答案的相关性和上下文的相关性。 **关键能力：**

RGB 的研究分析了不同大语言模型在处理 RAG 所需的四项基本能力方面的表现，包 括抗噪声能力、拒绝无效回答能力、信息综合能力和反事实稳健性，从而为检索增强型生成 设立了标准。

4 RAG 有哪些评估框架？

在 RAG 评估框架领域，RAGAS 和 ARES 是较新的方法。

4.1 RAGAS

RAGAS 是一个基于简单手写提示的评估框架，通过这些提示全自动地衡量答案的准确性、 相关性和上下文相关性。

算法原理：

1 答案忠实度评估：利用大语言模型（LLM） 分解答案为多个陈述，检验每个陈述与上下文 的一致性。最终，根据支持的陈述数量与总陈述数量的比例，计算出一个“忠实度得分”。 **2 答案相关性评估：**使用大语言模型（LLM）创造可能的问题，并分析这些问题与原始问题 的相似度。答案相关性得分是通过计算所有生成问题与原始问题相似度的平均值来得出的。 **3 上下文相关性评估：**运用大语言模型（LLM） 筛选出直接与问题相关的句子，以这些句子 占上下文总句子数量的比例来确定上下文相关性得分。

4.2 ARES

ARES 的目标是自动化评价 RAG 系统在上下文相关性、答案忠实度和答案相关性三个方 面的性能。ARES 减少了评估成本，通过使用少量的手动标注数据和合成数据，并应用预测 驱动推理（PDR） 提供统计置信区间，提高了评估的准确性。

算法原理：

生成合成数据集：ARES 首先使用语言模型从目标语料库中的文档生成合成问题和答案，创 建正负两种样本。

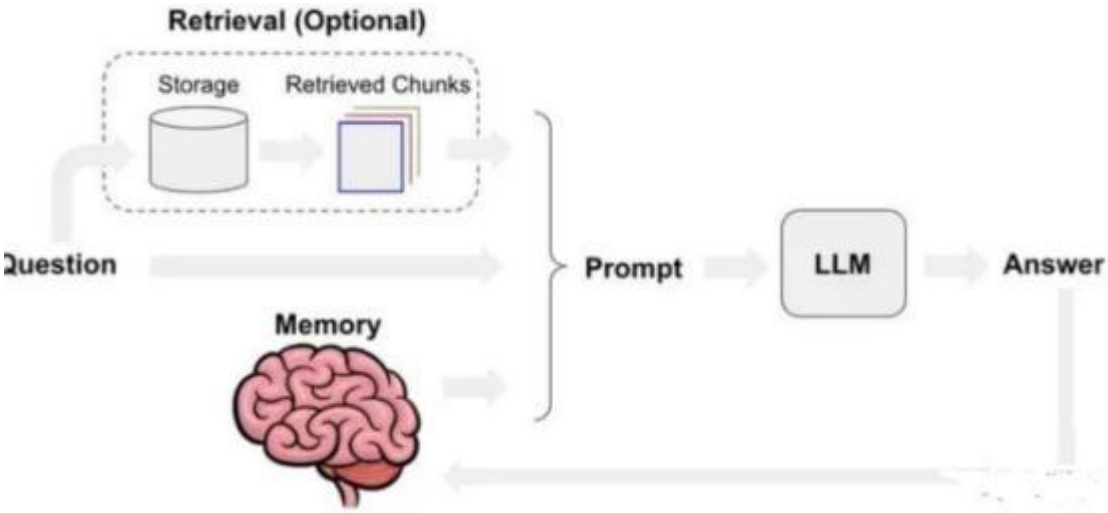
训练大语言模型（LLM） 裁判：然后，ARES 对轻量级语言模型进行微调，利用合成数据集 训练它们以评估上下文相关性、答案忠实度和答案相关性。

基于置信区间对RAG 系统排名：最后，ARES 使用这些裁判模型为 RAG 系统打分，并结 合手动标注的验证集，采用 PPI 方法生成置信区间，从而可靠地评估 RAG 系统的性能。

检索增强生成(RAG)优化策略篇

1 RAG 工作流程

从RAG 的工作流程看，RAG 模块有：文档块切分、文本嵌入模型、提示工程、大模型生成。



2 RAG 各模块有哪些优化策略?

文档块切分：设置适当的块间重叠、多粒度文档块切分、基于语义的文档切分、文档块摘要。

文本嵌入模型：基于新语料微调嵌入模型、动态表征。

提示工程优化：优化模板增加提示词约束、提示词改写。

大模型迭代：基于正反馈微调模型、量化感知训练、提供大context window 的推理模型。 此外，还可对query

召回的文档块集合进行处理，如：元数据过滤、重排序减少文档块数量。

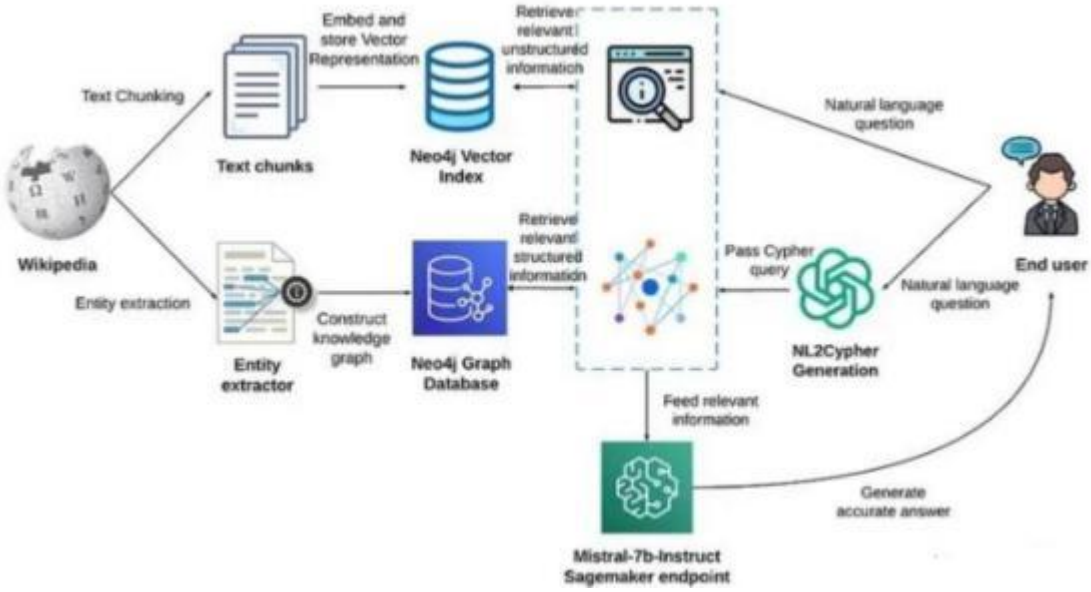
3 RAG 架构优化有哪些优化策略?

3.1 如何利用 知识图谱（KG）进行上下文增强?

1：向量数据库进行上下文增强存在问题： 无法获取长程关联知识

信息密度低（尤其当LLM context window 较小时不友好） 2 ：利用知识图谱（KG）进行上下文增强的策略：

增加一路与向量库平行的KG（知识图谱）上下文增强策略。



具体方式：对于 用户query，通过利用 NL2Cypher 进行 KG 增强；

优化策略：常用 图采样技术来进行KG 上下文增强

处理方式：根据query 抽取实体，然后把实体作为种子节点对图进行采样（必要时，可把KG 中节点和 query 中实体先向量化，通过向量相似度设置种子节点），然后把获取的子图转换 成文本片段，从而达到上下文增强的效果。

3.2 Self-RAG：如何让大模型对召回结果进行筛选?

1 ：典型RAG 架构中，向量数据库存在问题：

经典的RAG 架构中（包括KG 进行上下文增强），对召回的上下文无差别地与query 进 行合并，然后访问大模型输出应答。但有时召回的上下文可能与 query 无关或者矛盾， 此时就应舍弃这个上下文，尤其当大模型上下文窗口较小时非常必要（目前4k 的窗口 比较常见）

2 ： Self-RAG 则是更加主动和智能的实现方式，主要步骤概括如下：

a) 判断是否需要额外检索事实性信息（retrieve on demand），仅当有需要时才召回；

b) 平行处理每个片段：生产prompt+ 一个片段的生成结果；

c) 使用反思字段，检查输出是否相关，选择最符合需要的片段；

d) 再重复检索；

e) 生成结果会引用相关片段，以及输出结果是否符合该片段，便于查证事实。

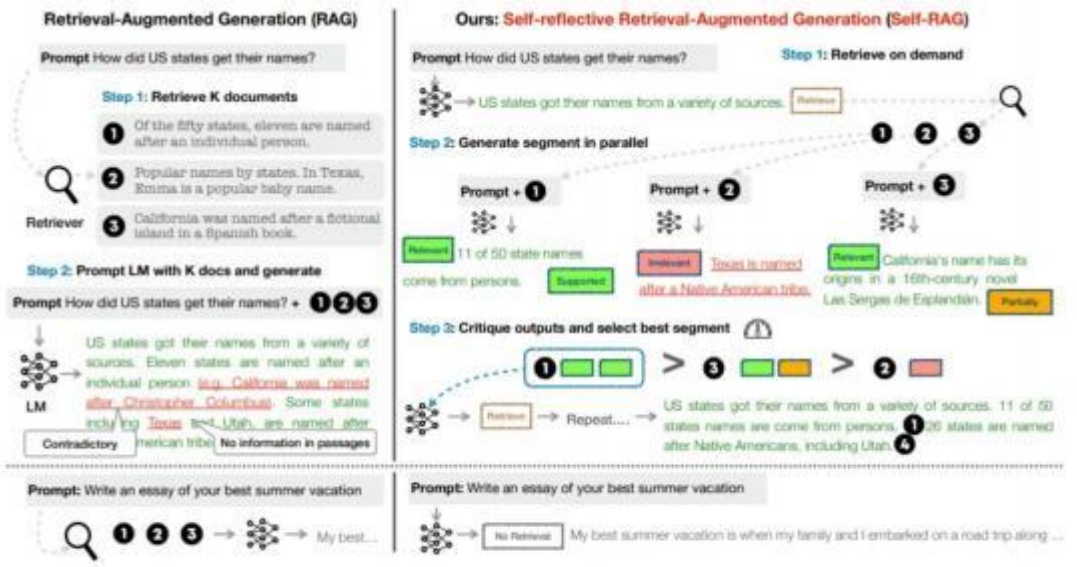


Figure 1: Overview of SELF-RAG. SELF-RAG learns to retrieve, critique, and generate text passages to enhance overall generation quality, factuality, and verifiability.

3：Self-RAG 的重要创新：Reflection tokens (反思字符)。通过生成反思字符这一特殊标记来 检查输出。这些字符会分为 Retrieve 和 Critique 两种类型，会标示：检查是否有检索的必 要，完成检索后检查输出的相关性、完整性、检索片段是否支持输出的观点。模型会基于原 有词库和反思字段来生成下一个 token。

4： Self-RAG 的训练过程?

对于训练，模型通过将反思字符集成到其词汇表中来学习生成带有反思字符的文本。 它是 在一个语料库上进行训练的，其中包含由 Critic 模型预测的检索到的段落和反思字符。 该 Critic 模型评估检索到的段落和任务输出的质量。 使用反思字符更新训练语料库，并训练 最终模型以在推理过程中独立生成这些字符。

为了训练 Critic 模型，手动标记反思字符的成本很高，于是作者使用 GPT-4 生成反思字符， 然后将这些知识提炼到内部 Critic 模型中。 不同的反思字符会通过少量演示来提示具体说 明。 例如，检索令牌会被提示判断外部文档是否会改善结果。

为了训练生成模型，使用检索和 Critic 模型来增强原始输出以模拟推理过程。 对于每个片 段，Critic 模型都会确定额外的段落是否会改善生成。如果是，则添加 Retrieve=Yes 标记，并检索前 K 个段落。然后 Critic 评估每段文章的相关性和支持性，并相应地附加标记。 最终通过输出反思字符进行增强。

然后使用标准的 next token 目标在此增强语料库上训练生成模型，预测目标输出和反思字 符。 在训练期间，检索到的文本块被屏蔽，并通过反思字符 Critique 和 Retrieve 扩展词汇 量。 这种方法比 PPO 等依赖单独奖励模型的其他方法更具成本效益。 Self-RAG 模型还 包含特殊令牌来控制和评估其自身的预测，从而实现更精细的输出生成。

5： Self-RAG 的推理过程?

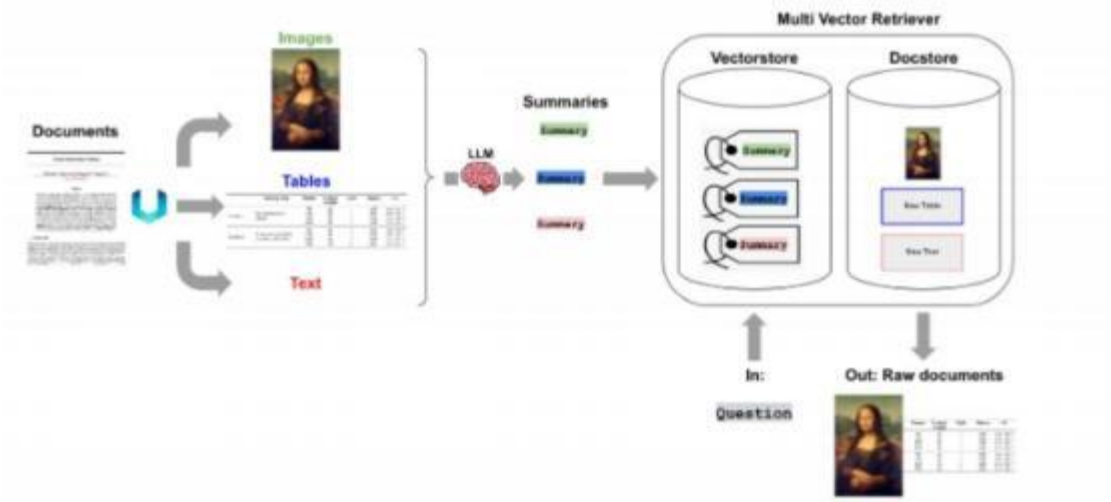
Self-RAG 使用反思字符来自我评估输出，使其在推理过程中具有适应性。根据任务的不同， 可以定制模型，通过检索更多段落来优先考虑事实准确性，或强调开放式任务的创造力。 该

模型可以决定何时检索段落或使用设定的阈值来触发检索。

当需要检索时，生成器同时处理多个段落，产生不同的候选。 进行片段级 beam search 以 获得最佳序列。 每个细分的分数使用 Critic 分数进行更新，该分数是每个批评标记类型的 归一化概率的加权和。可以在推理过程中调整这些权重以定制模型的行为。 与其他需要额 外训练才能改变行为的方法不同，Self-RAG 无需额外训练即可适应。

3.3 多向量检索器多模态 RAG 篇

多向量检索器（Multi-Vector Retriever）核心想法：将文档（用于答案合成）和引用（用于检 索）分离，这样可以针对不同的数据类型生成适合自然语言检索的摘要，同时保留原始的数 据内容。它可以与多模态 LLM 结合，实现跨模态的 RAG。



如何让 RAG 支持多模态数据格式?

3.3.1 如何让 RAG 支持半结构化 RAG（文本+表格）？

此模式要同时处理文本与表格数据。其核心流程梳理如下：

1 将原始文档进行版面分析（基于 Unstructured 工具),生成原始文本和原始表格。

2 原始文本和原始表格经summary LLM 处理，生成文本summary 和表格summary。

3 用同一个embedding 模型把文本summary 和表格summary 向量化，并存入多向量检 索器。

4 多向量检索器存入文本/表格 embedding 的同时，也会存入相应的 summary 和 raw data。

5 用户query 向量化后，用 ANN 检索召回raw text 和raw table。

6 根据query+raw text+raw table 构造完整prompt，访问 LLM 生成最终结果。

3.3.2 如何让 RAG 支持多模态 RAG（文本+表格+图片）？

对多模态RAG 而言有三种技术路线，如下我们做个简要说明：

选项 1： 对文本和表格生成summary，然后应用多模态embedding 模型把文本/表格 summary、原始图片转化成 embedding 存入多向量检索器。对话时，根据query 召回 原始文本/表格/图像。然后将其喂给多模态LLM 生成应答结果。

选项2： 首先应用多模态大模型（GPT4-V、LLaVA、FUYU-8b）生成图片summary 。 然后对文本/表格/图片 summary 进行向量化存入多向量检索器中。当生成应答的多 模态大模型不具备时，可根据query 召回原始文本/表格+图片summary。

选项3： 前置阶段同选项2 相同。对话时，根据query 召回原始文本/表格/图片。构 造完整Prompt ，访问多模态大模型生成应答结果。

3.3.3 如何让 RAG 支持私有化多模态 RAG（文本+表格+图片）？

如果数据安全是重要考量，那就需要把 RAG 流水线进行本地部署。比如可用 LLaVA-7b 生成图片摘要，Chroma 作为向量数据库，Nomic's GPT4All 作为开源嵌入模 型，多向量检索器，Ollama.ai 中的LLaMA2-13b-chat 用于生成应答。

3.4 RAG Fusion 优化策略

思路:

检索增强这一块主要借鉴了 RAG Fusion 技术，这个技术原理比较简单，概括起来 就是，当接收用户query 时，让大模型生成5-10 个相似的query，然后每个query 去 匹配5-10 个文本块，接着对所有返回的文本块再做个倒序融合排序，如果有需求就 再加个精排，最后取Top K 个文本块拼接至prompt。

优点:

增加了相关文本块的召回率;

对用户的query 自动进行了文本纠错、分解长句等功能

缺点:

无法从根本上解决理解用户意图的问题

3.5 模块化 RAG 优化策略

3.6 RAG 新模式优化策略

RAG 的组织方法具有高度灵活性，能够根据特定问题的上下文，对 RAG 流程中的模块进 行替换或重新配置。在基础的 Naive RAG 中，包含了检索和生成这两个核心模块，这个框 架因而具备了高度的适应性和多样性。目前的研究主要围绕两种组织模式：

增加或替换模块在增加或替换模块的策略中，我们保留了原有的检索 - 阅读结构，同时加 入新模块以增强特定功能。RRR 提出了一种重写 - 检索 - 阅读的流程，其中利用大语言模 型（LLM）的性能作为强化学习中重写模块的奖励机制。这样，重写模块可以调整检索查询， 从而提高阅读器在后续任务中的表现。

调整模块间的工作流程在调整模块间流程的领域，重点在于加强语言模型与检索模型之间 的互动。

3.7 RAG 结合 SFT

RA-DIT 方法策略：

1 更新 LLM 。以最大限度地提高在给定检索增强指令的情况下正确答案的概率；

2 更新检索器。以最大限度地减少文档与查询在语义上相似（相关）的程度。

优点：通过这种方式，使 LLM 更好地利用相关背景知识，并训练 LLM 即使在检索错误块 的情况下也能产生准确的预测，从而使模型能够依赖自己的知识。

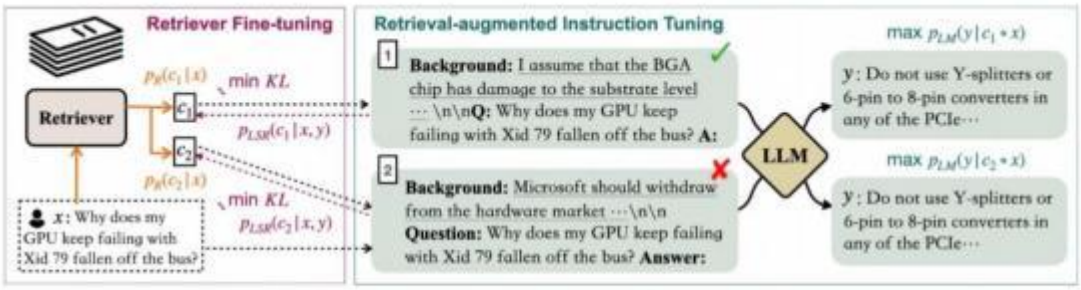


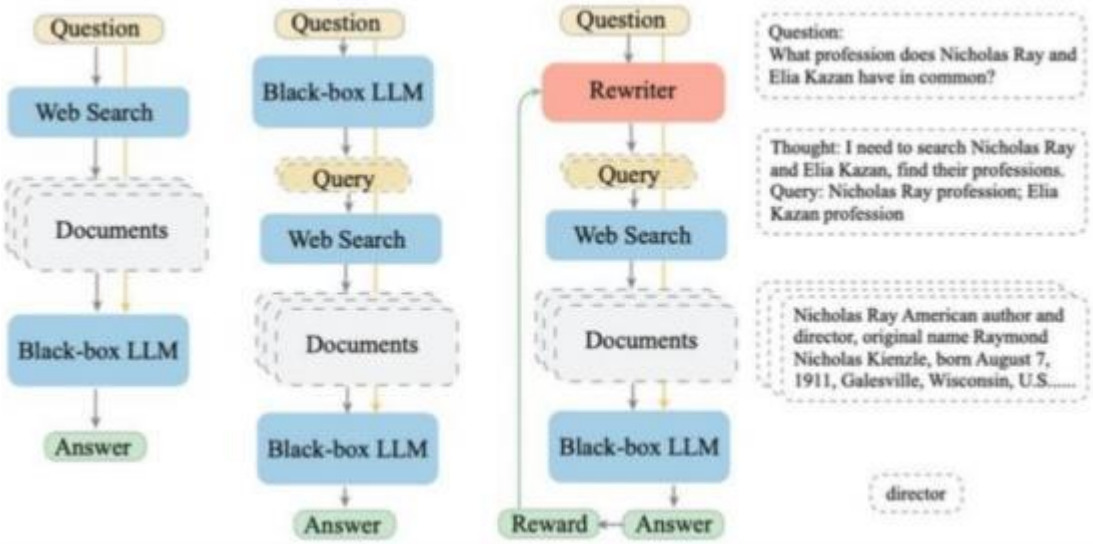
Figure 1: The RA-DIT approach separately fine-tunes the LLM and the retriever. For a given example, the LM-ft component updates the LLM to maximize the likelihood of the correct answer given the retrieval-augmented instructions (§2.3); the R-ft component updates the retriever to minimize the KL-Divergence between the retriever score distribution and the LLM preference (§2.4)

3.8 查询转换（Query Transformations）

动机：在某些情况下，用户的 query 可能出现表述不清、需求复杂、内容无关等问题；

查询转换（Query Transformations）：利用了大型语言模型(LLM)的强大能力，通过某种提 示或方法将原始的用户问题转换或重写为更合适的、能够更准确地返回所需结果的查询。 LLM 的能力确保了转换后的查询更有可能从文档或数据中获取相关和准确的答案。

查询转换的核心思想：用户的原始查询可能不总是最适合检索的，所以我们需要某种方式来 改进或扩展它。



3.9 Bert 在RAG 中具体是起到了一个什么作用?

RAG 中，对于一些传统任务（eg：分类、抽取等）用 bert 效率会快很多，虽然会牺牲一点 点效果，但是 比起推理时间，前者更被容忍；而对于一些生成式任务（改写、摘要等）， 必须得用 LLMs，原因：1. Bert 窗口有限，只支持 512 个字符，对于这些生成任务是远远 不够的；2.LLMs 生成能力比 Bert 系列要强很多，这个时候，时间换性能就变得很有意义。

4 RAG 索引优化有哪些优化策略?

4.1 嵌入优化策略

1 微调嵌入

影响因素：影响到 RAG 的有效性；

目的：让检索到的内容与查询之间的相关性更加紧密；

作用：可以比作在语音生成前对“听觉”进行调整，优化检索内容对最终输出的影响。 特别是在处理不断变化或罕见术语的专业领域，这些定制化的嵌入方法能够显著提高检 索的相关性。

2 动态嵌入（Dynamic Embedding）

介绍：不同于静态嵌入（static embedding),动态嵌入根据单词出现的上下文进行调整， 为每个单词提供不同的向量表示。例如，在 Transformer 模型（如 BERT）中，同一单 词根据周围词汇的不同，其嵌入也会有所变化。

3 检索后处理流程

动机:

② 一次性向大语言模型展示所有相关文档可能会超出其处理的上下文窗口限制。

②将多个文档拼接成一个冗长的检索提示不仅效率低，还会引入噪声，影响大语言模型 聚焦关键信息。

优化方法:

a) ReRank（重新排序）

b) Prompt 压缩
c) RAG 管道优化
d) 混合搜索的探索
e) 递归检索与查询引擎
f) StepBack-prompt 方法
g) 子查询
h) HyDE 方法

4.2 RAG 检索召回率低的解决方案

补充：尝试过不同大小的 chunk 和混合检索。效果都不太好，如何优化？ 个人排查方式：

1 知识库里面是否有对应答案？如果没有那就是知识库覆盖不全问题
2 知识库有，但是没召回：
q1：知识库知识是否被分割掉，导致召回出错，解决方法 修改分割方式 or 利用 bert 进行上下句预测，保证知识点完整性
q2：知识没有被召回，分析 query 和 doc 的特点：字相关还是语义相关，一般建议是先用 es 做召回，然后才用模型做精排

4.3 RAG 如何优化索引结构?

构建 RAG 时，块大小是一个关键参数。它决定了我们从向量存储中检索的文档的长度。 小块可能导致文档缺失一些关键信息，而大块可能引入无关的噪音。

找到最佳块大小是要找到正确的平衡。如何高效地做到这一点？试错法（反复验证）。 然而，这并不是让你对

每一次尝试进行一些随机猜测，并对每一次经验进行定性评估。

你可以通过在测试集上运行评估并计算指标来找到最佳块大小。LlamaIndex 有一些功 能可以做到这一点。可以在他们的博客中了解更多。

4.4 如何通过混合检索提升 RAG 效果?

虽然向量搜索有助于检索与给定查询相关的语义相关块，但有时在匹配特定关键词方面 缺乏精度。根据用例，有时可能需要精确匹配。

想象一下，搜索包含数百万电子商务产品的矢量数据库，对于查询“阿迪达斯参考 XYZ 运动鞋白色”，最上面的结果包括白色阿迪达斯运动鞋，但没有一个与确切的 XYZ 参考相 匹配。

相信大家都不能接受这个结果。为了解决这个问题，混合检索是一种解决方案。该策略 利用了矢量搜索和关键词搜索等不同检索技术的优势，并将它们智能地结合起来。

通过这种混合方法，您仍然可以匹配相关关键字，同时保持对查询意图的控制。

4.5 如何通过重新排名提升 RAG 效果?

当查询向量存储时，前K 个结果不一定按最相关的方式排序。当然，它们都是相关的， 但在这些相关块中，最相关的块可能是第 5 或第7 个，而不是第 1 或第2 个。

这就是重新排名的用武之地。

重新排名的简单概念是将最相关的信息重新定位到提示的边缘，这一概念已在各种框架 中成功实现，包括 LlamaIndex、LangChain 和 HayStack。

例如， Diversity Ranker 专注于根据文档的多样性进行重新排序， 而 LostInTheMiddleRanker 在上下文窗口的开始和结束之间交替放置最佳文档

5 RAG 索引数据优化有哪些优化策略?

5.1 RAG 如何提升索引数据的质量?

索引的数据决定了 RAG 答案的质量，因此首要任务是在摄取数据之前尽可能对其进 行整理。 (垃圾输入，垃圾输出仍然适用于此)

通过删除重复/冗余信息，识别不相关的文档，检查事实的准确性（如果可能的话）来实 现这一点。

使用过程中，对 RAG 的维护也很重要，还需要添加机制来更新过时的文档。

在构建 RAG 时，清理数据是一个经常被忽视的步骤，因为我们通常倾向于倒入所有文 档而不验证它们的质量。以下我建议可以快速解决一些问题：

通过清理特殊字符、奇怪的编码、不必要的 HTML 标签来消除文本噪音……还记得使 用正则表达式的老的 NLP 技术吗？可以把他们重复使用起来；

通过实施一些主题提取、降维技术和数据可视化，发现与主题无关的文档，删除它们； 通过使用相似性度量删除冗余文档。

5.2 如何通过添加元数据 提升 RAG 效果?

将元数据与索引向量结合使用有助于更好地构建它们，同时提高搜索相关性。 以下是一些元数据有用的情景：

--如果你搜索的项目中，时间是一个维度，你可以根据日期元数据进行排序

--如果你搜索科学论文，并且你事先知道你要找的信息总是位于特定部分，比如实验部分，你可以将文章部分添加为每个块的元数据并对其进行过滤仅匹配实验 元数据很有用，因为它在向量搜索之上增加了一层结构化搜索。

5.3 如何通过输入查询与文档对齐提升 RAG 效果?

LLMs 和 RAGs 之所以强大，因为它们可以灵活地用自然语言表达查询，从而降低数 据探索和更复杂任务的进入门槛。

然而，有时，用户用几个词或短句的形式作为输入查询，查询结果会出现与文档之间存 在不一致的情况。

通过一个例子来理解这一点。这是关于马达引擎的段落（来源：ChatGPT）

发动机堪称工程奇迹，以其复杂的设计和机械性能驱动着无数的车辆和机械。其核心是， 发动机通过一系列精确协调的燃烧事件将燃料转化为机械能。这个过程涉及活塞、曲轴和复 杂的阀门网络的同步运动，所有这些都经过仔细校准，以优化效率和功率输出。现代发动机 有多种类型，例如内燃机和电动机，每种都有其独特的优点和应用。对创新的不懈追求不断 增强汽车发动机技术，突破性能、燃油效率和环境可持续性的界限。无论是在开阔的道路上为汽车提供动力还是驱动工业机械，电机仍然是现代世界动态运动的驱动力。

在这个例子中，我们制定一个简单的查询，“你能简要介绍一下马达引擎的工作原理吗？”，与段落的余弦相似性为 0.72。 其实已经不错了，但还能做得更好吗?

为了做到这一点，我们将不再通过段落的嵌入来索引该段落，而是通过其回答的问题的 嵌入来索引该段落。考虑这三个问题，段落分别回答了这些问题：

--发动机的基本功能是什么?

--发动机如何将燃料转化为机械能?

--发动机运行涉及哪些关键部件，它们如何提高发动机的效率? 通过计算得出，它们与输入查询的相似性分别为：

0.864 0.841 0.845

这些值更高，表明输入查询与问题匹配得更精确。

将块与它们回答的问题一起索引，略微改变了问题，但有助于解决对齐问题并提高搜索 相关性：我们不是优化与文档的相似性，而是优化与底层问题的相似性。

5.4 如何通过提示压缩提升 RAG 效果?

研究表明，在检索上下文中的噪声会对 RAG 性能产生不利影响，更精确地说，对由 LLM 生成的答案产生不利影响。

一些方案建议在检索后再应用一个后处理步骤，以压缩无关上下文，突出重要段落，并 减少总体上下文长度。

选择性上下文等方法和 LLMLingua 使用小型LLM 来计算即时互信息或困惑度，从而 估计元素重要性。

5.5 如何通过 查询重写和扩展 提升RAG 效果?

当用户与 RAG 交互时，查询结果不一定能获得最佳的回答，并且不能充分表达与向量 存储中的文档匹配的结果。为了解决这个问题，在送到 RAG 之前，我们先发生给 LLM 重 写此查询。这可以通过添加中间 LLM 调用轻松实现，但需要继续了解其他的技术实现（参 考论文《Query Expansion by Prompting Large Language Models》）。

6 RAG 未来发展方向

RAG 的三大未来发展方向：垂直优化、横向扩展以及 RAG 生态系统的构建。

6.1 Rag 的垂直优化

尽管 RAG 技术在过去一年里取得了显著进展，但其垂直领域仍有几个重点问题有待深入探究

RAG 中长上下文的处理问题 RAG 的鲁棒性研究

RAG 与微调（Fine-tuning）的协同作用 RAG 的工程应用

在工程实践中，诸如如何在大规模知识库场景中提高检索效率和文档召回率，以及如何保障 企业数据安全——例如防止 LLM 被诱导泄露文档的来源、元数据或其他敏感信息——都 是亟待解决的关键问题。

6.2 RAG 的水平扩展

在水平领域，RAG 的研究也在迅速扩展。从最初的文本问答领域出发，RAG 的应用逐渐拓 展到更多模态数据，包括图像、代码、结构化知识、音视频等。

6.3 RAG 生态系统

下游任务和评估

通过整合来自广泛知识库的相关信息，RAG 展示了在处理复杂查询和生成信息丰富回 应方面的巨大潜力。

众多研究表明，RAG 在开放式问题回答、事实验证等多种下游任务中表现优异。RAG 模型不仅提升了下游应用中信息的准确性和相关性，还增加了回应的多样性和深度。

RAG 的成功为其在多领域应用的适用性和普适性的探索铺平了道路，未来的工作将围 绕此进行。特别是在医学、法律和教育等专业领域的知识问答中，RAG 的应用可能会相比 微调（fine-tuning） 提供更低的训练成本和更优的性能表现。

同时，完善 RAG 的评估体系，以更好地评估和优化它在不同下游任务中的应用，对提 高模型在特定任务中的效率和效益至关重要。这涉及为各种下游任务开发更精准的评估指标 和框架，如上下文相关性、内容创新性和无害性等。

此外，增强 RAG 模型的可解释性，让用户更清楚地理解模型如何以及为何作出特定反 应，也是一项重要任务

技术栈

在 RAG 的技术生态系统中，相关技术栈的发展起着推动作用。例如，随着 ChatGPT 的流行，LangChain 和 LLamaIndex 迅速成为知名技术，它们提供丰富的 RAG 相关 API， 成为大模型时代的关键技术之一。

与此同时，新型技术栈也在不断涌现。尽管这些新技术并不像 LangChain 和 LLamaIndex 那样功能众多，但它们更注重自身的独特特性。例如，FlowiseAI6 着重于低代 码操作，使用户能够通过简单的拖拽实现 RAG 代表的各类 AI 应用。其他新兴技术如 HayStack 、Meltno 和 Cohere Coral技术栈的发展与 RAG 的进步相互促进。新技术对现有技术栈提出了更高的要求，而技 术栈功能的优化又进一步推动了 RAG 技术的发展。综合来看，RAG 工具链的技术栈已经 初步建立，许多企业级应用逐步出现。然而，一个全面的一体化平台仍在完善中。

五、大模型（LLMs）显存问题面

1. 大模型大概有多大，模型文件有多大?

一般放出来的模型文件都是 fp16 的，假设是一个 nB 的模型，那么模型文件占 2nG，fp16 加载到显存里做推理也是占 2n G，对外的pr 都是 10n 亿参数的模型。

2. 能否用4 * v100 32G 训练vicuna 65b?

不能。

首先，llama 65b 的权重需要5* v100 32G 才能完整加载到GPU。

其次，vicuna 使用flash-attention 加速训练，暂不支持v100，需要turing 架构之后的显卡。（fastchat 上可以通过调用train 脚本训练vicuna 而非train_mem，其实也是可以训练的）

3. 如果就是想要试试65b 模型，但是显存不多怎么办?

最少大概50g 显存，可以在llama-65b-int4（gptq）模型基础上LoRA[6]，当然各种库要安装 定制版本的。

4. nB 模型推理需要多少显存?

考虑模型参数都是fp16，2nG 的显存能把模型加载。

5. nB 模型训练需要多少显存?

基础显存：模型参数+梯度+优化器，总共16nG。

activation 占用显存，和max len、batch size 有关。

解释：优化器部分必须用fp32（乎乎fp16 会导致训练不稳定），所以应该是2+2+12=16，参 考ZeRO 论文。

注：以上算数不够直观，举个例子?

7B 的vicuna 在fsdp 下总共 160G 显存勉强可以训练。（按照上面计算7*16=112G 是基础显 存）所以全量训练准备显存 20nG 大概是最低要求，除非内存充足，显存不够 offload 内存 补。

6. 如何估算模型所需的RAM?

首先，我们需要了解如何根据参数量估计模型大致所需的 RAM，这在实践中有很重要的参 考意义。我们需要通过估算设置 batch_size，设置模型精度，选择微调方法和参数分布方法 等。接下来，我们用LLaMA-6B 模型为例估算其大致需要的内存。

首先考虑精度对所需内存的影响：

fp32 精度，一个参数需要 32 bits, 4 bytes. fp16 精度，一个参数需要 16 bits, 2 bytes. int8 精度，一个参数需要 8 bits, 1 byte.

其次，考虑模型需要的 RAM 大致分三个部分：

模型参数 梯度

优化器参数

模型参数：等于参数量*每个参数所需内存。

对于 fp32，LLaMA-6B 需要 6B*4 bytes = 24GB 内存

对于 int8，LLaMA-6B 需要 6B*1 byte = 6GB 梯度：同上，等于参数量*每个梯度参数所需内存。 优化器参数：不同的优化器所储存的参数量不同。

对于常用的 AdamW 来说，需要储存两倍的模型参数（用来储存一阶和二阶 momentum）。fp32 的 LLaMA-6B，AdamW 需要 6B*8 bytes = 48 GB int8 的 LLaMA-6B，AdamW 需要 6B*2 bytes = 12 GB

除此之外，CUDA kernel 也会占据一些 RAM，大概 1.3GB 左右，查看方式如下。

```
> torch.ones((1 , 1)).to("cuda") >
print_gpu_utilization()
> > >
GPU memory occupied: 1343 MB
```

综上，int8 精度的 LLaMA-6B 模型部分大致需要 6GB+6GB+12GB+1.3GB = 25.3GB 左右。 再根据LLaMA 的架构（hidden_size = 4096, intermediate_size =11008, num_hidden_layers = 32, context_length = 2048）计算中间变量内存。

每个 instance 需要：

(4096 +11008)* 2048 *32 * 1byte = 990MB

所以一张 A100（80GBRAM）大概可以在 int8 精度；batch_size = 50 的设定下进行全参数 训练。查看消费级显卡的内存和算力：

2023 GPU Benchmark and Graphics Card Comparison Chart

https://www.gpucheck.com/gpu-benchmark-graphics-card-comparison-chart

7. 如何评估你的显卡利用率

zero3 如果没有 nvlink，多卡训练下会变慢。但是一直不知道究竟会变得多慢，下面给出几 种方法来评估自己在训练时发挥了多少gpu 性能，以及具体测试方法。

7.1 flops 比值法

测试工具：deepspeed

参考数据: nvidia 公布的显卡fp16 峰值计算速度 (tensor core) gpu 利用率 = 实

测的 flops/显卡理论上的峰值flops

举例：deepspeed 实测flops 100tflops，而用的是A100 卡理论峰值312tflops，可以得到GPU 利用率只有32.05%

7.2 throughout 估计法

测试工具：手动估算 或者 deepspeed

参考数据：论文中的训练速度或者吞吐量

吞吐量 = example 数量/秒/GPU * max_length

gpu 利用率 = 实际吞吐量 / 论文中的吞吐量（假设利用率 100%） 举例:

实测训练时处理样本速度为 3 example/s，一共有4 卡，max length 2048，则吞吐量为 1536 token/s/gpu

根据llama 论文知道，他们训练 7B 模型的吞吐量约为 3300 token/s/gpu，那么GPU 利用率 只有46.54%

7.3 torch profiler 分析法

测试工具：torch profiler 及 tensorboard 参考数据：无

利用torch profiler 记录各个函数的时间，将结果在tensorboard 上展示，在gpu kenel 视图下， 可以看到tensor core 的利用率，比如30%

总结：以上三种方法，在笔者的实验中能得到差不多的利用率指标。

从准确性上看，方案三 > 方案一 > 方案二 从易用性上看，方

案二 > 方案一 > 方案三

如果不想改代码就用方案二估算自己的训练速度是不是合理的，如果想精确分析训练速度的 瓶颈还是建议使用方案三。

8. 测试你的显卡利用率实现细节篇

8.1 如何查看多机训练时的网速?

iftop 命令，看网速很方便。

8.2 如何查看服务器上的多卡之间的NVLINK topo?

```
$ nvidia-smitopo -m
```

8.3 如何查看服务器上显卡的具体型号?

```
cd /usr/local/cuda/samples/1_Utilities/deviceQuery make
./deviceQuery
```

8.4 如何查看训练时的flops? （也就是每秒的计算量）

理论上，如果 flops 比较低，说明没有发挥出显卡的性能。

如果基于 deepspeed 训练，可以通过配置文件很方便的测试。 {

```
"flops_profiler": {
    "enabled": true,
    "profile_step": 1,
    "module_depth": -1, "top_modules": 1,
    "detailed": true,
    "output_file": null
    }}
```

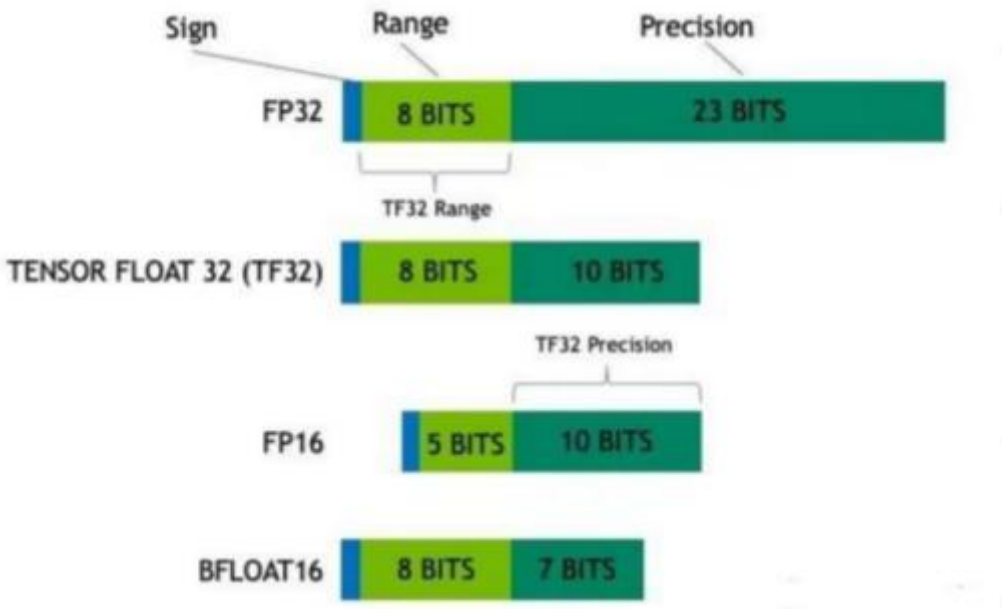
参考：https://www.deepspeed.ai/tutorials/flops-profiler/

8.5 如何查看对deepspeed 的环境配置是否正确?

```
$ ds_report
```

8.6 tf32 格式有多长?

19 位



8.7 哪里看各类显卡算力比较?

https://lambdalabs.com/gpu-benchmarks

8.8 (torch profiler) 如何查看自己的训练中通信开销?

用pytorch profiler 查看，下面给出基于transformers 的一种快捷的修改方式。

https://github.com/yqhu/profiler-workshop/blob/c8d4a7c30a61cc7b909d89f88f5fd36b70c55769/hf_training_trainer_prof.py

用记录的pt.trace.json 文件放到tensorboard 上，可以看出tensor core 的利用率。

根据实践经验，使用deepspeed zero3 时，pcie 版本的卡很大部分时间都在通信上，AllGather 和ReduceScatter 的时间超过tensor core 计算的时间，所以flops 上不去。

六、大模型（LLMs）增量预训练篇

1. 为什么要增量预训练?

有一种观点，预训练学知识，指令微调学格式，强化学习对齐人类偏好，LIMA 等论文算是 这一观点的证据。所以要想大模型有领域知识，得增量预训练。（靠指令微调记知识不靠谱， 不是几十w 条数据能做到的。）

2. 进行增量预训练 需要做哪些准备工作?

模型底座选型

主流是LLaMA，因为scaling 法则，可能LLaMA 做了充分预训练。（当然有版权问题） 这里备选BLOOM，感觉基座比LLaMA 差，但是也有7B 版本。

Falcon、CPM-bee、Aquila、Baichuan 待实验，license 友好，但生态和效果都是问题。其实， 因为结构上都类似LLaMA，未来估计会出现整合这些模型的项目。

（Falcon 公布的训练语料中没有中文）这里没列ChatGLM 和ChatGLM2，因为有种说法在 SFT 模型上增量预训练效果比较差。（未证实）

数据收集

这里最经典的开源预训练数据还是wudao 的200G 和thepile 这两个数据集(怀念一下Open- Llama)加起来有 1T 的文本量，足够前期玩耍了。

其实，刚开始实践的时候，不需要太多样本，先收集GB 量级的领域文本跑通流程即可。

数据清洗

当然这里数据治理可能是 chatgpt 魔法的最关键的部分，最基础的是把网页爬取数据中的广 告清理掉。Falcon 论文里介绍了数据清洗的手段，对于我们很有参考意义。

3. 增量预训练 所用训练框架?

超大规模训练

如果是真大规模炼丹，那没什么好说的，直接 3D 并行。

Megatron-Deepspeed 拥有多个成功案例，炼LLaMA 可以参考LydiaXiaohongLi 大佬的实现。 （实在太强）

https://github.com/microsoft/Megatron-DeepSpeed/pull/139

炼 BLOOM 可以直接找到Bigscience 的git 仓库。然而，转 checkpoint

还是挺费劲的。

少量节点训练

小门小户一共就几台机器几张卡的话，3D 并行有点屠龙术了。

张量并行只有在nvlink 环境下才会起正向作用，但提升也不会太明显。 可以分2 种情况：

单节点或者多节点（节点间通信快）：直接deepspeed ZeRO 吧。（笔者用了linly 的增量预训 练代码，但有能力的最好用其他代码）比如，Open-Llama 的fork 版本。

https://github.com/RapidAI/Open-Llama

多节点（但节点间通信慢）:考虑用流水线并行，参考另一个大佬的实现。

https://github.com/HuangLK/transpeeder

少量卡训练

如果资源特别少，显存怎么也不够，可以上 LoRA。

https://github.com/shibing624/MedicalGPT

4. 增量预训练 训练流程 是怎么样?

数据预处理

参考LLaMA 的预训练长度，也把数据处理成2048 长度（如果不够，做补全） 这里要吐槽，

tencentpretrain 数据处理脚本的默认长度竟然是 128。

分词器

有很多工作加LLaMA 中文词表，但是考虑到没有定论说加中文词表会更好，先用原版的吧， 500k 的tokenizer.model。

https://github.com/ymcui/Chinese-LLaMA-Alpaca

原始模型

可以使用一个中文增量预训练后的版本，当然这里坑挺大的，各家框架的模型层名不太一样。 为了快速跑通，用脚本快速转一下，能成功加载就行。

训练参数

如果显存不够，可以zero3+offload。其他参数暂时默认吧。（事实上没有想象中慢） 多机的话可以配一下deepspeed 的hostfile。

观测训练进展

这一点可能是最重要的，跑通只是第一步，根据训练情况反复调整比较重要。

可以使用wandb，记录loss ，flops，吞吐速度，已消耗的token 数，和测试ppl。

模型转换

不同框架的checkpoint 格式不同，还会根据并行度分成很多个文件。 以ZeRO 为例，我的转换流程（很挫）是：

zero to f32 f32 to fp16

fp16 to huggingface 格式

模型测试

转为标准huggingface 格式后可以用各种支持llama 的前端加载，比如text-generation-webui。 可以简单测试下续写能力，验证下模型是否正常。

至此，我们获得了 1 个增量预训练过的大模型基座

5. 增量预训练一般需要多大数据量?

首先要确保你有足够大量的数据集，至少有几B 的token；否则几十条数据的情况我更推荐 模型微调。

6. 增量预训练过程中，loss 上升正常么?

通常增量预训练开始的阶段会出现一段时间的loss 上升，随后慢慢收敛。

7. 增量预训练过程中，lr 如何设置?

学习率是一个很重要的参数，因为 lr 的大小会出现以下问题：

如果lr 过大，那loss 值收敛会更困难，旧能力损失的会更大； 如果lr 过小，那可能难以学到新知识。

当你数据集比较小（例如 100B 以下?），那建议使用较小的学习率。例如可以使用pre-train 阶段最大学习率的 10%。通常7B 模型pre-train 阶段的学习率大概是3e-4，所以我们可以选 择3e-5。

并且需要根据你的batch size 做相应缩放。通常lr 缩放倍数为batch size 倍数的开方。例如 batch size 增大4倍，学习率对应扩大2 倍即可。

8. 增量预训练过程中，warmup_ratio 如何设置?

warmup_ratio 也很重要。通常LLM 训练的warmup_ratio 是epoch * 1%左右。例如pre-train 阶段一般只训一个epoch，则 ratio 是0.01 ； SFT 通常3 个epoch ， ratio 对应为0.03。

但是如果做CPT，建议warmup_ratio 调大一点。如果你的数据集很大，有几百b，那warmup 其实不影响最重的模型效果。但通常我们的数据集不会有那么大，所以更小的 ratio 可以让 模型 “过渡” 得更平滑。

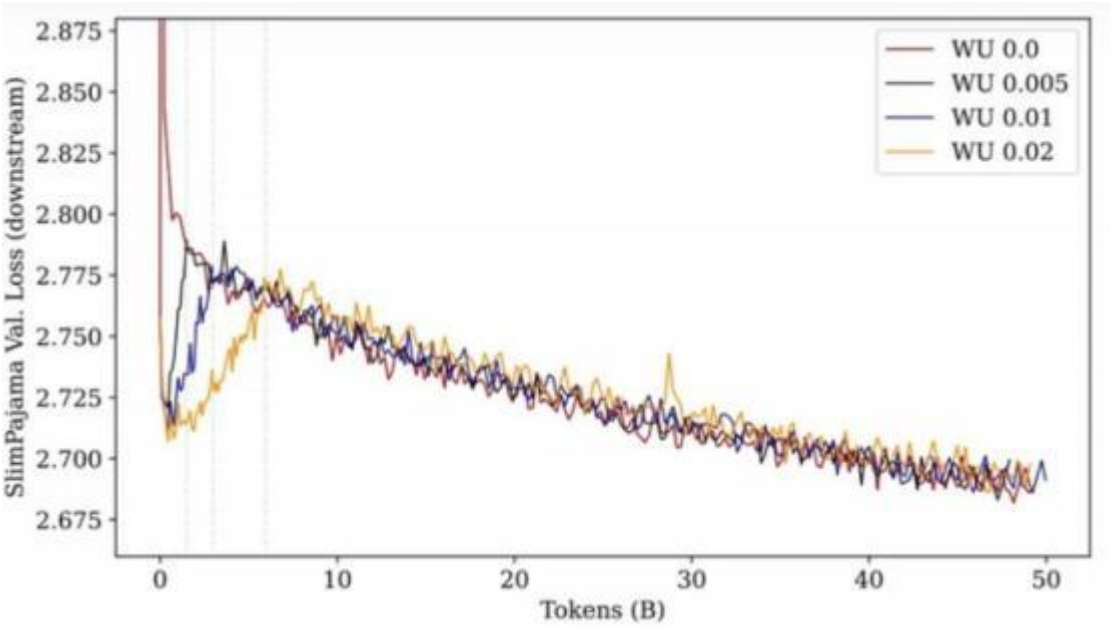
学习率和warmup_ratio 是两个相辅相成的概念，二者通常是成正比的关系。或者说如果你正 在用一个较大的学习率，那你或许可以同时尝试增加warmup 来防止模型 “烂掉” 。

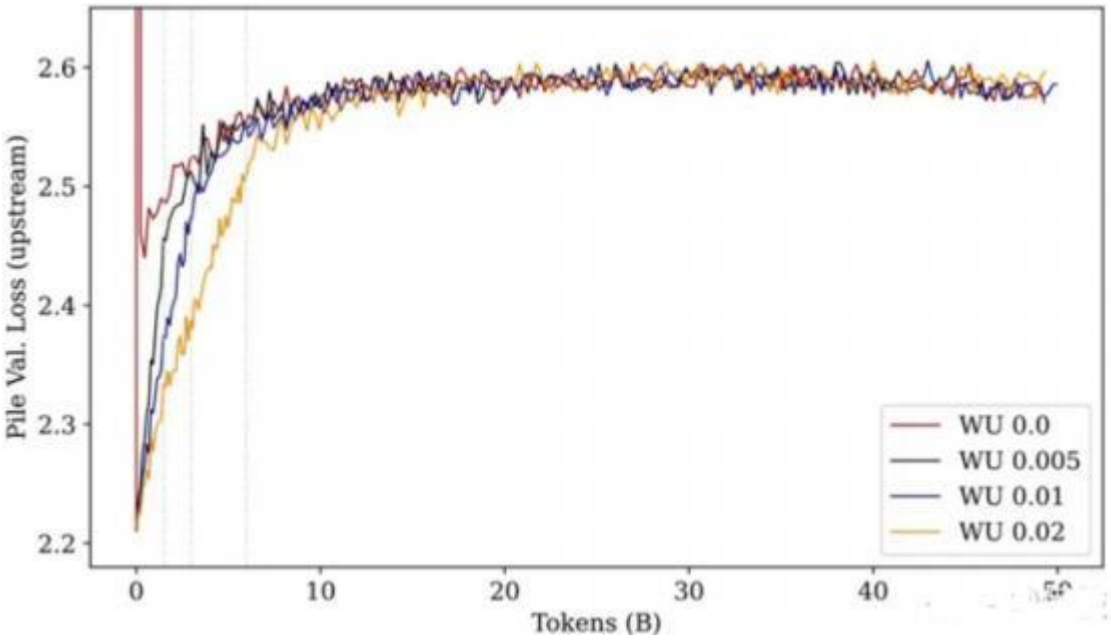
9.warmup 的步数对大模型继续预训练是否有影响?

warmup 介绍：warmup 是一种 finetune 中常用的策略，指学习率从一个很小的值慢慢上升 到最大值;

对比实验设计：使用 不同 4 种不同预热步数（eg：0%, 0.5%, 1%, 2%）来进行实验

。 不同预热百分比步数的性能图，上图为下游任务 loss，下图为上游任务 loss





实验结果：当模型经过「充分」训练后，不管多长的预热步数最后的性能都差不多。

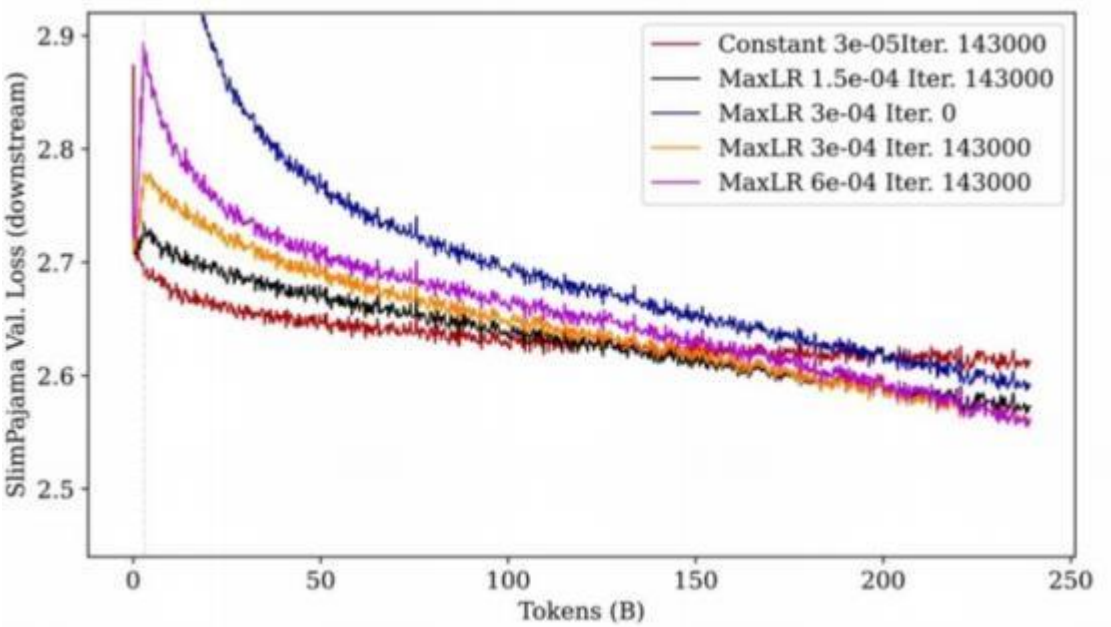
注：但，这种前提是「充分训练」，如果只看训练前期的话，使用更长的预热步数（黄色的 线），无论是「上游任务」还是「下游任务」，模型的 Loss 都要比其他预热步数要低（下游 学的快，上游忘的慢）。

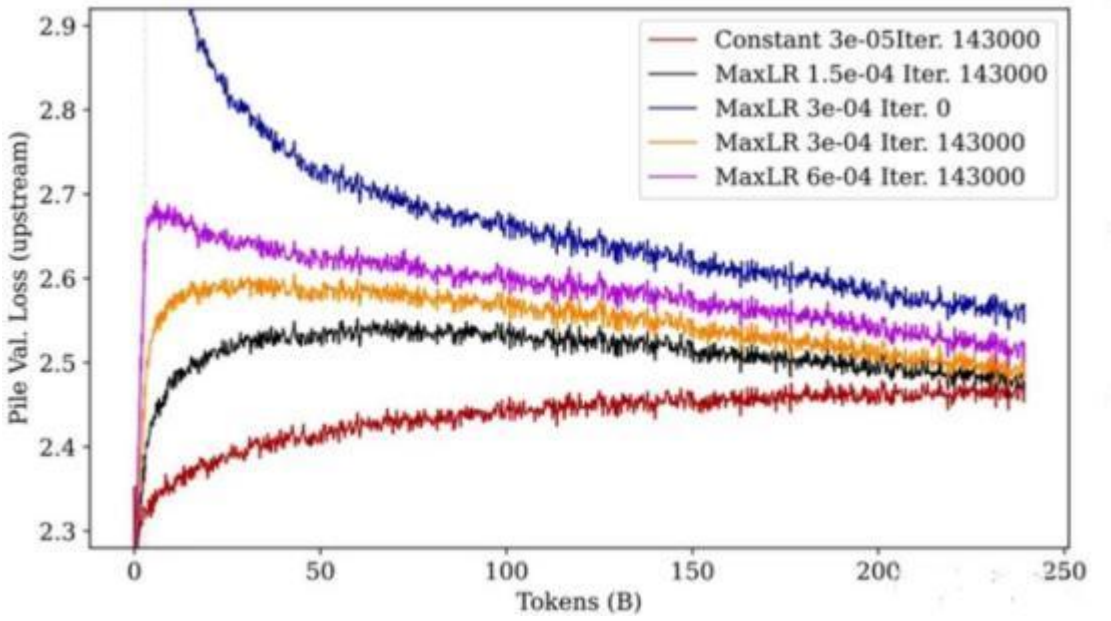
10. 学习率大小对大模型继续预训练后上下游任务影响?

对比实验：使用了 4 种不同的最大学习率进行对比实验 **实验结论:**

经过充分训练后，学习率越大（紫色），下游性能最好，上游性能最差（忘得最多）。

未经过预训练的模型（蓝色）无论是上游任务还是下游任务，都不如预训练过的模型效果。 注：前期训练，尽管紫色线条在最后的 loss 是最低的，但在前期 loss 会增加的非常大，随 后下降。

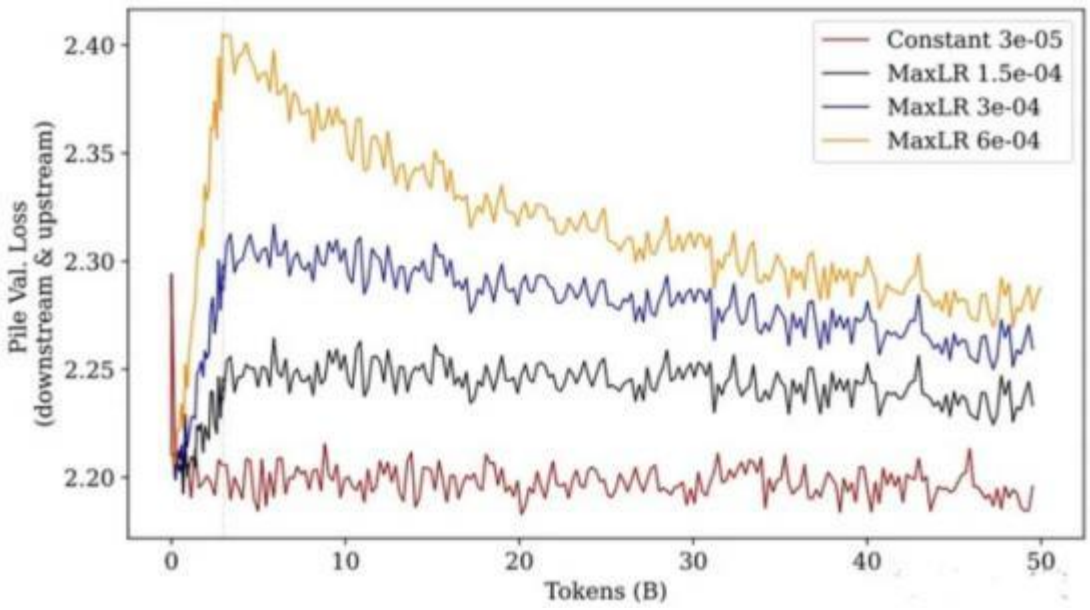




解释一下这里为什么这么关注训练前期，是因为在真实训练中，我们可能不一定会增强图中 所示的 250B 这么多的 tokens，尤其是在模型参数很大的情况中。所以，当资源不允许充分 训练的情况下，较小的学习率和较长的 warmup 步数可能是一个不错的选择。

11. 在初始预训练中使用 Rewarmup 对大模型继续预训练性能影响?

对比实验：不切换数据集，而是继续在之前的「预训练数据集（The Pile）」上继续训练：



实验结果：无论使用多大学习率的 warmup 策略，效果都不如使用常量学习率。

这进一步证明，在原数据集上使用 warmup 接着训练会造成性能损伤，学习率越大则损伤 越大，且这种损伤是无法在后续的训练中被找回的。

注：PS：这里提示我们，当预训练中遇到了训练中断需要继续训练时，我们应该在重新开始 训练时将学习率恢复到中断之前的状态（无论是数值还是衰减率）。

七、大模型蒸馏篇

1. 一、知识蒸馏和无监督样本训练?
2. 二、对知识蒸馏知道多少，有哪些改进用到了?
3. 三、谈一下对模型量化的了解?
4. 四、模型压缩和加速的方法有哪些?
5. 五、你了解的知识蒸馏模型有哪些?

一、知识蒸馏和无监督样本训练?

知识蒸馏是利用大模型把一个大模型的知识压缩到一个小模型上。具体来说你在一个训练集上得到了一 个非常好的较大的模型，

然后你把这个模型冻结，作为Teacher模型也叫监督模型，然后你再造一个较小参数的模型叫做Student 模型，我们的目标就是利用冻结的Teacher模型去训练Student模型。

A.离线蒸馏： Student在训练集上的loss和与Teacher模型的loss作为总的loss ， 一起优化。

B.半监督蒸馏： 向Teacher模型输入一些input得到标签， 然后把input和标签传给Student模型

还有个自监督蒸馏，直接不要Teacher模型，在最后几轮epoch，把前面训练好的模型作为Teacher进行 监督。

目前知识蒸馏的一个常见应用就是对齐ChatGPT。

然后这个无监督样本训练，我看不懂意思。如果是传统的无监督学习，那就是聚类，主成分分析等操 作。如果是指知识蒸馏的话，就是离线蒸馏的方式，只不过损失只有和Teacher的loss。

二、对知识蒸馏知道多少，有哪些改进用到了?

知识蒸馏是一种通过将一个复杂模型的知识转移到一个简单模型来提高简单模型性能的方法。这种方法 已经被广泛应用于各种深度学习任务中。其中一些改进包括：

1. 使用不同类型的损失函数和温度参数来获得更好的知识蒸馏效果。
2. 引入额外的信息来提高蒸馏的效果，例如将相似性约束添加到模型训练中。
3. 将蒸馏方法与其他技术结合使用，例如使用多任务学习和迁移学习来进一步改进知识蒸馏的效果。

三、谈一下对模型量化的了解?

模型量化是一种将浮点型参数转换为定点型参数的技术，以减少模型的存储和计算复杂度。常见的模型 量化方法包括：

1. 量化权重和激活值，将它们转换为整数或小数。
2. 使用更小的数据类型，例如8位整数、 16位浮点数等。
3. 使用压缩算法，例如Huffman编码、可逆压缩算法等。

模型量化可以减少模型的存储空间和内存占用，同时也可以加速模型的推理速度。但是，模型量化可能 会对模型的精度造成一定的影响，因此需要仔细权衡精度和计算效率之间的平衡。

四、模型压缩和加速的方法有哪些?

参数剪枝（Parameter Pruning）：删除模型中冗余的参数，减少模型的大小。通常情况下，只有很少 一部分参数对模型的性能贡献较大，其余参数对性能的贡献较小或没有贡献，因此可以删除这些冗余参 数。

量化（Quantization）：将浮点型参数转换为更小的整数或定点数，从而减小模型大小和内存占用，提 高计算效率。

知识蒸馏（Knowledge Distillation）：利用一个较大、较准确的模型的预测结果来指导一个较小、较简 单的模型学习。这种方法可以减小模型的复杂度，提高模型的泛化能力和推理速度。

网络剪枝（Network Pruning）：删除模型中冗余的神经元，从而减小模型的大小。与参数剪枝不同， 网络剪枝可以删除神经元而不会删除对应的参数。

蒸馏对抗网络（Distillation Adversarial Networks）：在知识蒸馏的基础上，通过对抗训练来提高模型的鲁棒性和抗干扰能力。

模型量化（Model Quantization）：将模型的权重和激活函数的精度从32位浮点数减少到更小的位数， 从而减小模型的大小和计算开销。

层次化剪枝（Layer-wise Pruning）：对模型的不同层进行不同程度的剪枝，以实现更高效的模型压缩 和加速。

低秩分解（Low-Rank Decomposition）：通过将一个较大的权重矩阵分解为几个较小的权重矩阵，从 而减少计算开销。

卷积分解（Convolution Decomposition）：将卷积层分解成几个更小的卷积层或全连接层，以减小计算开销。

网络剪裁（Network Trimming）：通过对模型中一些不重要的连接进行剪裁，从而减小计算开销。

五、你了解的知识蒸馏模型有哪些?

FitNets：使用一个大型模型作为教师模型来指导一个小型模型的训练。

Hinton蒸馏：使用一个大型模型的输出作为标签来指导一个小型模型的训练。

Born-Again Network（ BAN）：使用一个已经训练好的模型来初始化一个新模型，然后使用少量的数据重新训练模型。

TinyBERT：使用一个大型BERT模型作为教师模型来指导一个小型BERT模型的训练。