推荐收藏！大模型常考面试题总结（含答案）

1.RAG 技术体系的总体思路

数据预处理->分块（这一步骤很关键，有时候也决定了模型的效果）->文本向量化->query 向量化->向量检索->重排->query+检索内容输入 LLM->输出

2.使用外挂知识库主要为了解决什么问题

- 克服遗忘问题
- 提升回答的准确性、权威性、时效性
- 解决通用模型针对一些小众领域没有涉猎的问题
- 提高可控性和可解释性，提高模型的可信度和安全性

3. 如何评价 RAG 项目效果的好坏

针对检索环节的评估：

MMR 平均倒排率：查询（或推荐请求）的排名倒数

Hits Rate 命中率：前 k 项中，包含正确信息的项的数目占比

NDCG

针对生成环节的评估：

非量化：完整性、正确性、相关性

量化：Rouge-L

4. 大模型的幻觉问题、复读机问题是什么

幻觉问题：即生成的内容是无意义的或不忠实于提供的源内容

复读机问题：重复生成某些话

5. 针对问题 4，有没有什么解决办法

针对幻觉问题：引入外挂知识库，加入一些纠偏规则，限制输出长度等

针对复读机问题：

丰富数据集的多样性，预处理时尽量过滤重复无意义的文本

同义词替换等做数据增强

温度参数调整

后处理与过滤

6. 出现问题 4 的原因有哪些

针对幻觉问题：幻觉问题主要分为两大类，一类是生成结果与数据源不一致，自相矛盾。另一类是用户问题超出了大模型的认知。针对前者可能是训练数据和源数据不一致、数据没对齐或者编码器理解能力的缺陷和解码器策略错误可能导致幻觉。后者则是用户的问题不在语言模型认知范围内。

针对复读机问题：数据质量不高，存在大量单一、重复文本，文本过长（补充：当你的前面的条件文本过长时，大模型的输出的几个短文本会被原始的很长的条件文本淹没，继续预测下一个 token 的话，在模型看起来可能条件仍然是差不多的，此时如果使用 greedy search，只选择概率最大的一个 token，模型极大可能会将前面已经生成的短文本重新预测成概率最大的文本，以此类推，会一直重复下去。）

7. 当前主流的开源大模型是哪个，其架构具体是怎样的?

当前开源影响范围最广，生态建设最好的开源大模型是 Meta 的 LLaMA。其依旧采用 Transformers 架构，并做了如下改动：

为了提高训练稳定性，对每个子层做输入前置归一化，归一化函数为 RMSNorm（受 GPT-3 启发）

为了提升性能，SwiGLU 激活函数替换 ReLU 激活函数（受 PaLM 启发）

从绝对位置嵌入，改为旋转嵌入（受 GPT-neo 启发）

使用 causal multi-head attention 的一个高效实现来减少内存占用和运行时间

8. 有哪几种 SFT 方法

全微调

Adapter Tuning

Prefix Tuning

Prompt Tuning

P-Tuning v1

lora

RLHF

（SFT 时，学习率设置为预训练阶段的 10%，一般会取得不错的效果）

9. 什么是 lora 微调

在原始 PLM (Pre-trained Language Model) 旁边增加一个旁路，做一个降维再升维的操作，来模拟所谓的 intrinsic rank。训练的时候固定 PLM 的参数，只训练降维矩阵 A 与升维矩阵 B。而模型的输入输出维度不变，输出时将 BA 与 PLM 的参数叠加。用随机高斯分布初始化 A ，用 0 矩阵初始化 B，保证训练的开始此旁路矩阵依然是 0 矩阵。

10. RAG 的检索阶段，常见的向量检索模型有哪些?

ANN 算法

乘积向量

暴力搜索

hnswlib

KD 树

11. 针对通用的 RAG，你觉得还有哪些改进点?

query 侧：做 query 的纠错、改写，规范化和扩展

对向量数据库做层次索引，提高检索效率和精度

对 LLM 模型微调，针对当前垂直领域引入知识库，提升回答的专业性、时效性和正确性

对最终输出做后处理，降低输出的不合理 case

12. 什么是 LangChain

LangChain 为大型语言模型提供了一种全新的搭建和集成方式，通过这个强大的框架，我们可以将复杂的技术任务简化，让创意和创新更加易于实现。有点类似于神经网络开发与 tensorflow/pytorch 之间的关系

13. LangChain 的常用模块有哪些?

document_loaders 文档加载

text_splitter 文档分块

embedding.huggingface 向量化

vectorstores 向量存储

chain.RetrievalQA 检索问答

14. SFT 和 RLHF 优劣对比

方法	优点	缺点
SFT	1. 启动简单，仅需要问答对数据就可以进行训练；	
2. 训练简单，直接对LLM进行调参，大部分情况下，训练几个epoch就可达到预期；		
3. 显存资源相比于RLHF来说耗费低	1. 模型的效果非常依赖于SFT数据的质量，天花板取决于SFT数据标注的质量；	
2. 对标注要求高，对一条指令来说，要标注很高质的回答其实是一件非常困难的事情，既耗时又耗力。		
RLHF	1. 直接利用人类偏好来进行指导大模型的训练，使大模型的输出更能满足人们的要求；	
2. 在安全性和事实性上，都能有很大的提升。	1. 训练消耗大量的显存；2. 训练的过程中很不稳定，想要成功的完成PPO的训练不是那么容易的事情；3. 除了SFT阶段的数据，还得标注Reward model的数据。在对齐人类偏好的过程中，标谁更好比较容易，但是想标谁比谁好多少，又差多少，这不是件容易的事情。be	

15. 详细介绍一下 RLHF

RLHF，全称为 Reinforcement Learning from Human Feedback（基于人类反馈的强化学习），是一种机器学习方法，特别适用于训练像语言模型这样的 AI 系统，以生成更符合人类期望和价值观的输出。这种方法结合了强化学习（RL）与直接的人类评估，旨在优化模型生成的内容，使其既高质量又符合社会规范。以下是 RLHF 的关键组成部分和工作流程：

16. 大模型训练经常出现一些 OOM 问题，在现有硬件基础下，有什么性能提升 trick

梯度累积

混合精度训练

减轻模型参数

分布式训练

减少批量大小

增加硬件资源

数据处理与加载优化：例如，可以使用数据流水线技术来并行加载和处理数据，减少内存中同时存在的数据量

17. LLaMA 模型输入句子理论上可以无限长吗?

不可以

受限于计算资源

训练阶段长句子会导致梯度消失或者梯度爆炸（因为它依赖前面的词进行最大似然估计作为损失函数，这个最大似然估计化简一下就是连乘的形式，容易造成梯度消失或者梯度爆炸）

推理阶段会增加预测错误率

18. 如何让大模型处理更长的文本?

分块处理，同时重叠保证连贯性

增加模型参数量，复杂化模型架构，提高对更长文本的捕捉与表达能力

19. 大模型推理时，显存中有那几部分数据?

模型参数

输入数据

计算中间结果

内存管理策略：某些深度学习框架在推理时采用了一种延迟释放显存的策略，即显存不会立即释放，而是保留一段时间以备后续使用。这种策略可以减少显存的分配和释放频率，提高推理效率，但也会导致显存一直占用的现象。

20. 介绍下 ChatGLM

首先要说起它的基座 GLM， GLM 既可以做 Encoder 也可以做 Decoder。

主要通过 两种 mask 方式来实现：

[mask]：bert 形式，随机 mask 文本中的短 span

[gmask]：gpt 形式，mask 末尾的长 span

在 chatglm 里面做生成任务时，是用 [gmask]。chaglm2 中完全采用 gmask 来进行预训练。

在 ChatGLM 的内部结构中的变换，从下到上依次是：

位置编码：从 BERT 的训练式位置编码转变为旋转位置编码

激活函数：从 BERT 中的 GeLU 转变为 GLU， 在 ChatGLM2 中又变成了 SwiGLU

LayerNormalization：采用的是 DeepNorm，是对 post-Normalization 的改进，即在残差之后做 Normalization。

在 ChatGLM 中，把 layer-normalization 改为 RMSNormalization。

在 ChatGLM 2.0 中还添加了一些其他变化：

FlashAttenion：利用显存和内存来做加速

Multi-Query Attention：多个头只采用一个 KV 对，通过参数共享来降低显存占用

21. 介绍下 GLU 激活函数和 SwiGLU 激活函数

GLU 的核心思想是通过门控机制来过滤信息，进而提高网络的表达能力和泛化能力。（门控机制有助于长距离建模）

SwishGLU 的核心思想是将 Swish 和 GLU 结合起来，SwishGLU 实际上只是多乘了一个 g(x)。

22. LLaMA1/2 的异同

首先从数据角度，lamma2.0 为 2.0T， lamma1 是 1.4T。其次是上下文长度， lamma1 是 2k，lamma2 是 4k。

最后从模型架构角度：

从整体上看，二者都遵循自回归 Transformer 的架构，只不过内部的细节有些不同。

位置编码：二者都采用了旋转位置编码

Normalization：二者都采用 pre-normalization，只不过 1.0 中采用原始的 LayerNormalization，2.0 中采用的是 RMSNorm

激活函数：采用 SwiGLU

23. 模型在训练和推理的时候各占用显存的多少?

一般来说，训练占用模型参数量的 16 倍大小（优化器、梯度、模型本身）；推理占用模型参数量的 2 倍大小（fp16 是一个单位参数大小对应两个字节，int8 是一个单位参数大小对应一个字节）。

24. 详细说说 Deepspeed 的机制

以下内容引用自该篇文章

是一种数据并行的方法，采用的 ring all reduce 方式。

传统的 parameter server 是 server 和 client 的方式，client 通过计算分配给自己的数据，产生梯度，传给 server，server 做聚合，然后把聚合后的参数再传给 client，这个方式的弊端是 server 容易成为瓶颈，server 通信量太大。另外可能一个 client 失败，会导致其他 client 等待。

Ring all reduce 是一种分布式的方式，各个节点分配通信量。总的通信量和 ps 没啥变化，但是通信的压力平摊到各个 GPU 上了，GPU 之间的通信可以并行进行。

假如，GPU 数量是 N，把模型参数分成 N 份，每个 GPU 要存放整个参数。每个 GPU 也要分配训练数据。当一次迭代，N 个 GPU 之间要经过一个 scatter 和 gather 操作，reduce-scatter 是将不同 gpu 上对应的参数的 gradient 相加，一共需要通讯（N-1）次。All-gather 是将合并完整的参数，传到其他 gpu 上，需要通讯（N-1）次。一次 all reduce，单卡通信量为 2*sita。

Zero 包括 3 种方案，逐步递进：

zero1：将 adam 的参数分割成 N 份，这样一个 GPU 上只能保存一份 adam 参数：这对于 forward 没啥影响，gradient 需要进行一次 all-reduce，但是只能更新一部分参数，所以 W 需要进行一次 all-gather，通信量为 3N*sita，存储为 12*sita/N + 4*sita

zero2: 将 adamw，gradient 都分割成 N 份，梯度就不需要 all-gather 了，只需要 scatter 了，w 需要 all-gather，通讯量为 2N*sita

zero3: 将参数，adam 和 gradient 都分割，forward 的时候，需要将 w all-gather，backfoward 时，还需要把 w all-gather 回来，计算梯度，丢掉不属于自己的 w，然后对梯度做 reduce scatter，更新 w，通讯量为 3N*sita。

最后采用采用 stage3：用 1.5 倍的通讯开销，换回近 120 倍的显存

另外，还有 ZeRO-Offload 是基于 Zero2，将 adam 和 gradient 放到内存中，在 cpu 内起了 N 个线程计算。其中的一条主线是 gradient 总是需要 scatter 的，感觉这个数据并行标志。这里需要注意一点 不管是 forward 还是 backward，都是需要有完整的 w 的。另外有了 gradient，以及 adamW 的参数，才能更新 W。

25. 什么是混合精度训练

FP64 用 8 个字节共 64 位，FP32 用 4 个字节共 32 位，FP16 用 2 个字节共 16 位。在神经网络模型的训练过程中，通常默认使用单精度（FP32）浮点数据类型来表示网络模型的权重和其他参数。

为什么需要混合精度训练呢?

使用 FP16 训练神经网络，相比使用 FP32 有以下优点。

内存占用减少：FP16 的位宽是 FP32 的一半，所以权重等参数占用的内存也减少了一半，这样就可以训练更大的网络模型或者使用更多的数据。

通讯效率提高：对于分布式训练，特别是大模型训练，通讯的开销是一个性能瓶颈，通讯的位宽减少了一半，就可以加快通讯速度，减少等待时间，提高数据流通效率。

计算效率更高：在一些专门的 AI 加速芯片上，使用 FP16 的计算性能比 FP32 更快。

但是使用 FP16 也会带来一些问题，主要有两个方面：

数据溢出和

舍入误差

为了利用 FP16 的优势，提高深度学习训练的效率和性能，同时避免精度溢出和舍入误差的影响，可以采用 FP16

和 FP32 的混合精度训练，主要涉及到一下三个重要技术点：

权重备份（Weight Backup）

损失放大（Loss Scaling）

精度累加（Precision Accumulated）

26. 什么是 prefix LLM 和 casual LLM

prefix LM：token 可以相互看到，输入双向注意力，输出单向注意力（ChatGLM、ChatGLM2、U-PaLM）

casualLM：严格自回归，从左到右的单向注意力 （LLaMA-7B、LLaMa 衍生物、Qwen）

27. 说一说针对 MHA 后续的一些计算优化工作

KV cache，核心思想：因为 transformers 是自回归，每一个 token 的预测包含大量前文的重复冗余计算。优化点是将需要重复利用的中间结果存下来，避免重复计算

MQA，核心思想：虽然 SRAM 的带宽很大，但是容量很小（A100 只有大概 20M），所以要进一步减少需要缓存的数据。MQA 中的多头只有 Q 是不一样的，K、V 完全相同。

GQA，核心思想：进一步减少需要缓存的数据大小，K、V 分组复制，数据大小介于 MQA 和 MHA 之间。

FlashAttention，核心思想：将 Q、K、V 切分为更小的块，从 HBM 中加载到 SRAM，需要计算的时候直接从 SARM 中读（因为目前 transformers 的计算效率瓶颈不在于计算速度，而是 IO）

FlashAttention 涉及到很多计算 trick，有空单独开一篇文章来说

28. 说说 attention 几种常见的计算方式

self-attention

din 的 attention 计算在得到权重之后，没有进行 softmax 归一化操作，而是保留了权重原始的信号强度。原始权重是通过网络最后一层激活函数得到，因此可以利用激活函数的特点，将其值限制在 0-1 之间，如 sigmoid 函数，从而使得到的原始权重值可直接用于加权求和，保留了权重的差异性。使用 softmax 操作虽然可以进行归一化，但同时会弱化了权重之间的差异性，有损用户兴趣的局部聚焦性。