【大模型 LLMs 基础面 Plus】

Layer normalization-方法篇

一、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

二、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。

三、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)
```

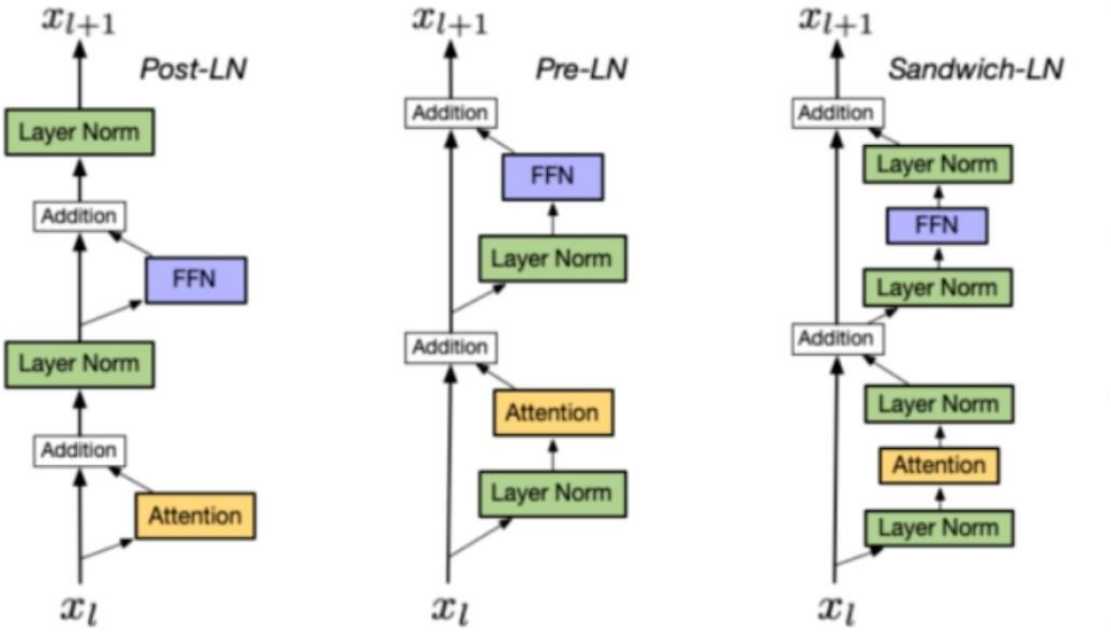
Deep Norm 有什么优点？

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： Post LN 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：

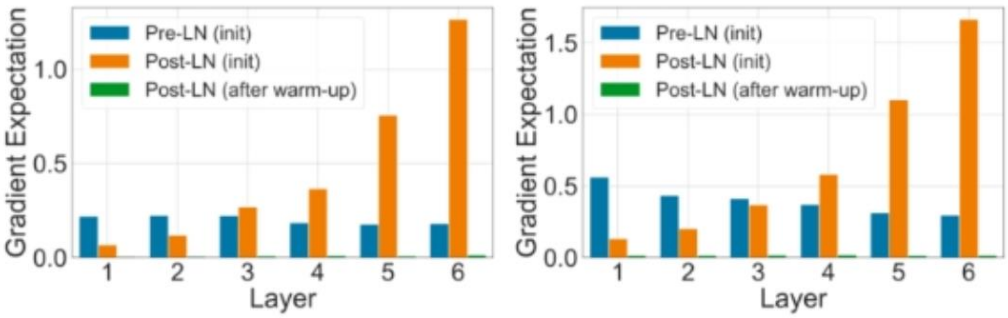
位置： layer norm 在残差链接中

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？

$$SwiGLU = Swish_\beta(xW) \otimes xV$$

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。

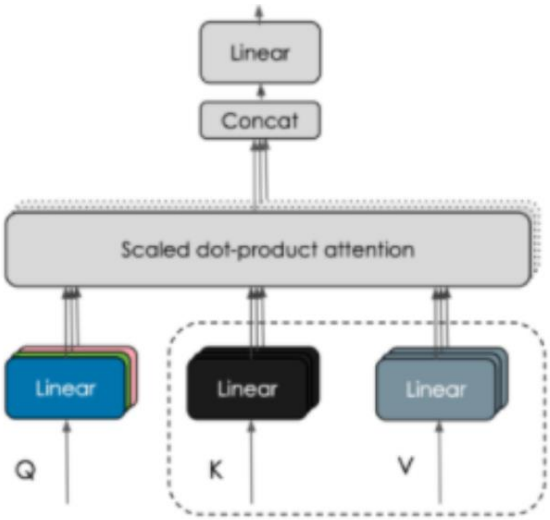
4 Multi-Query Attention 篇

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？

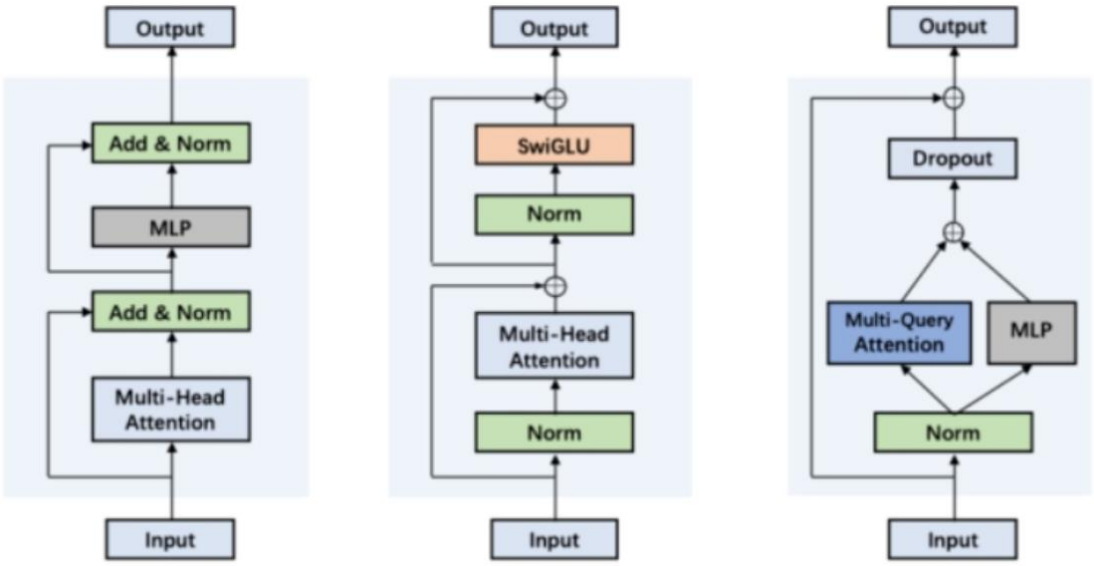
ChatGLM2，LLaMA2-34B/70B 使用了 Grouped query attention。

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。
-

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？

- **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。

- **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。

- **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。

- **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。

- **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）。

- **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）。

- **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

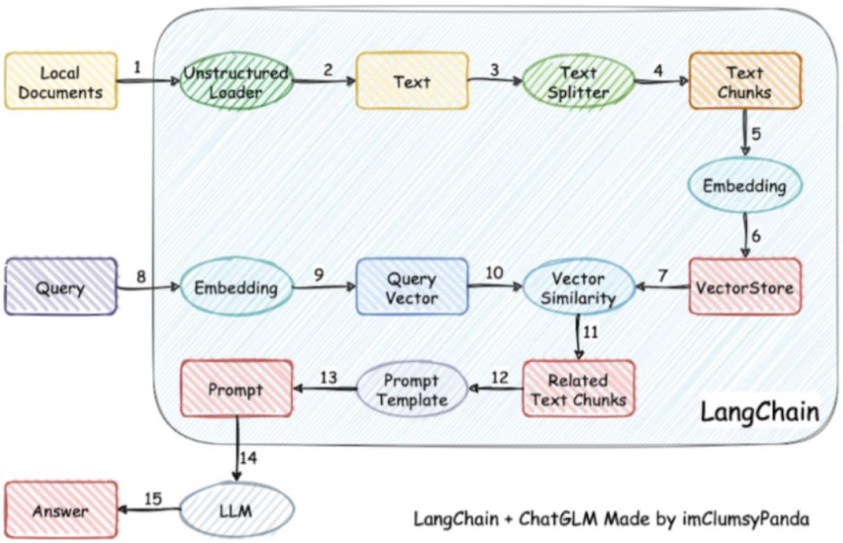
1.1 为什么大模型需要外挂 (向量) 知识库？

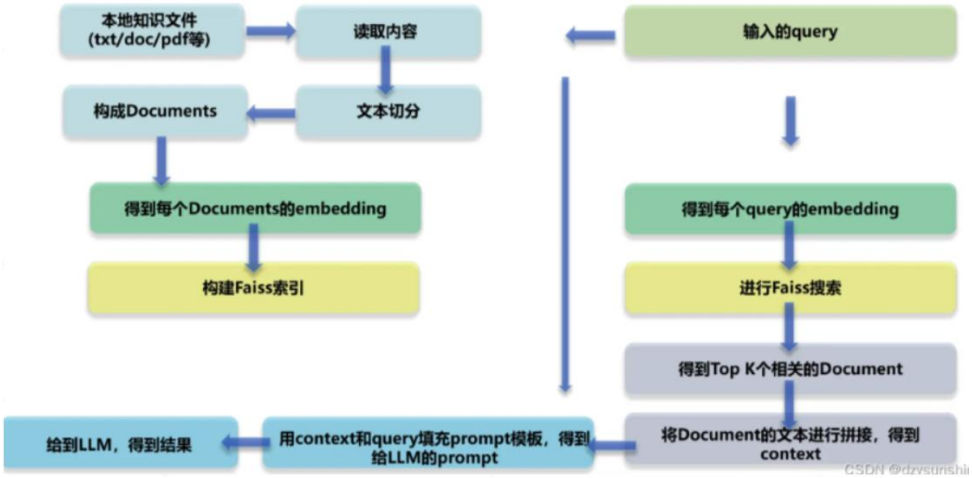
如何将外部知识注入大模型，最直接的方法：利用外部知识对大模型进行微调。

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）。

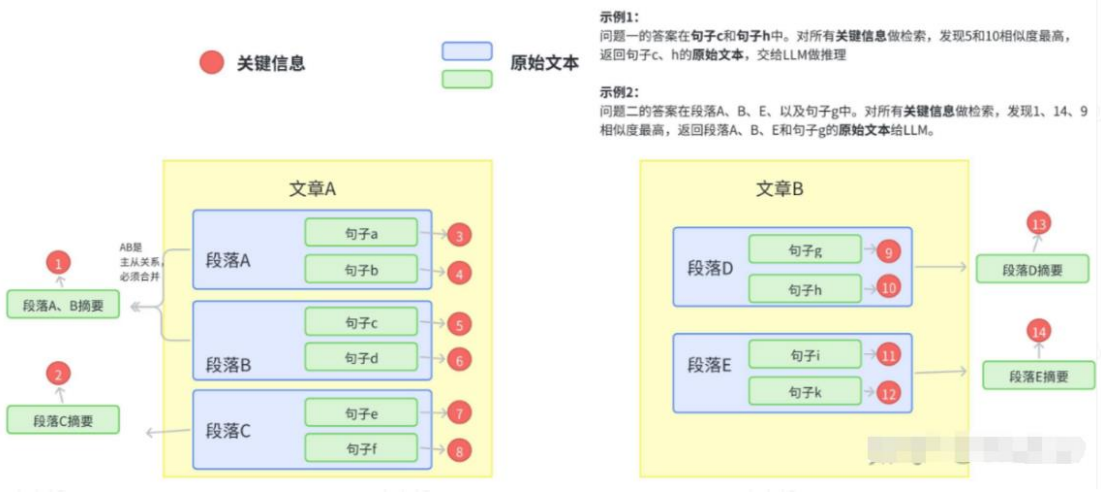
所以说，我们的文档切片最好是按照语义切割。

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

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。

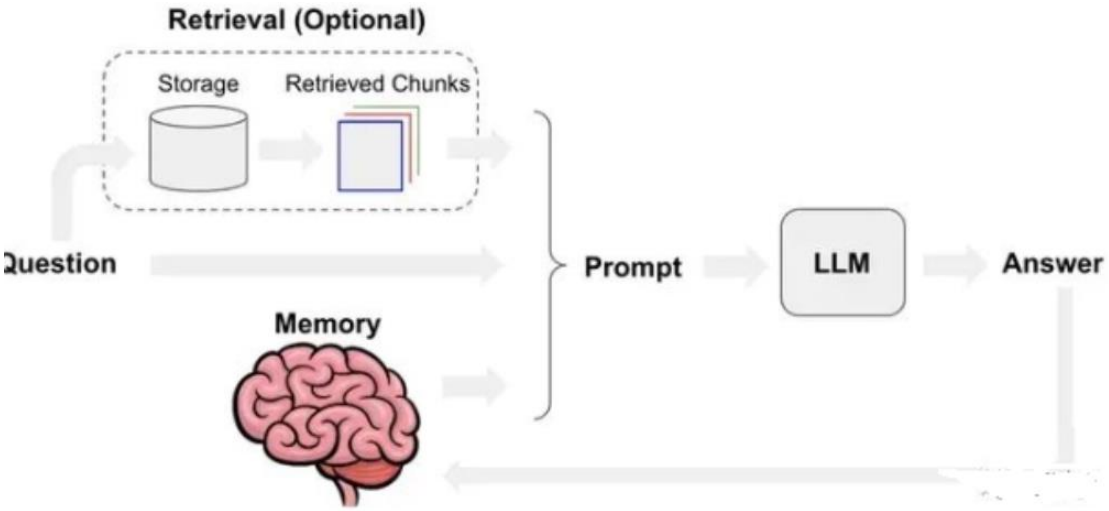
训练大语言模型 (LLM) 裁判：然后，ARES 对轻量级语言模型进行微调，利用合成数据集训练它们以评估上下文相关性、答案忠实度和答案相关性。

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？

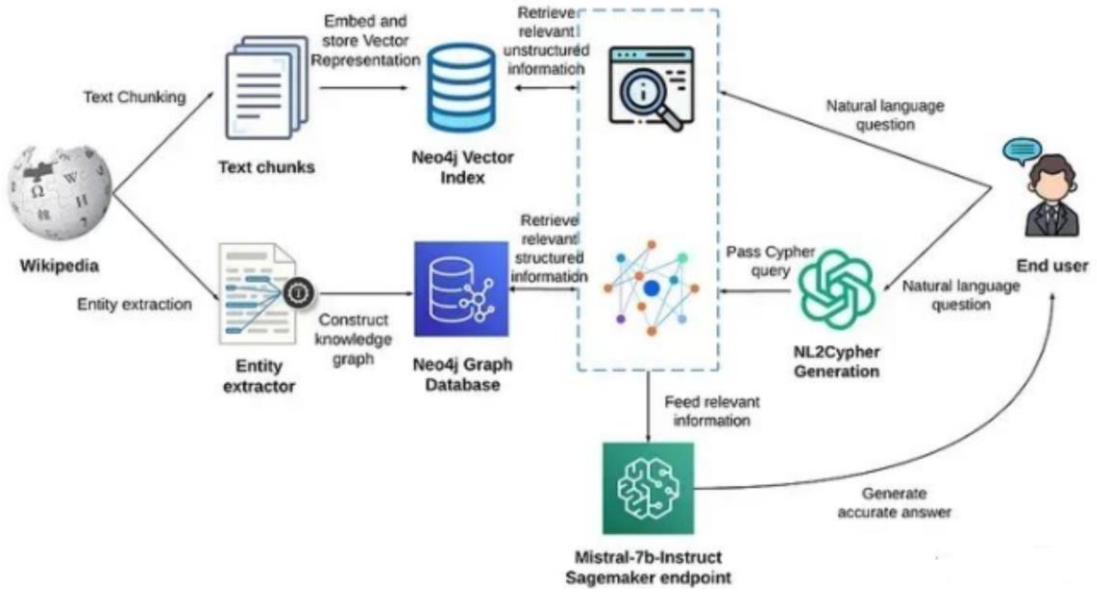
1：向量数据库进行上下文增强存在问题：

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。

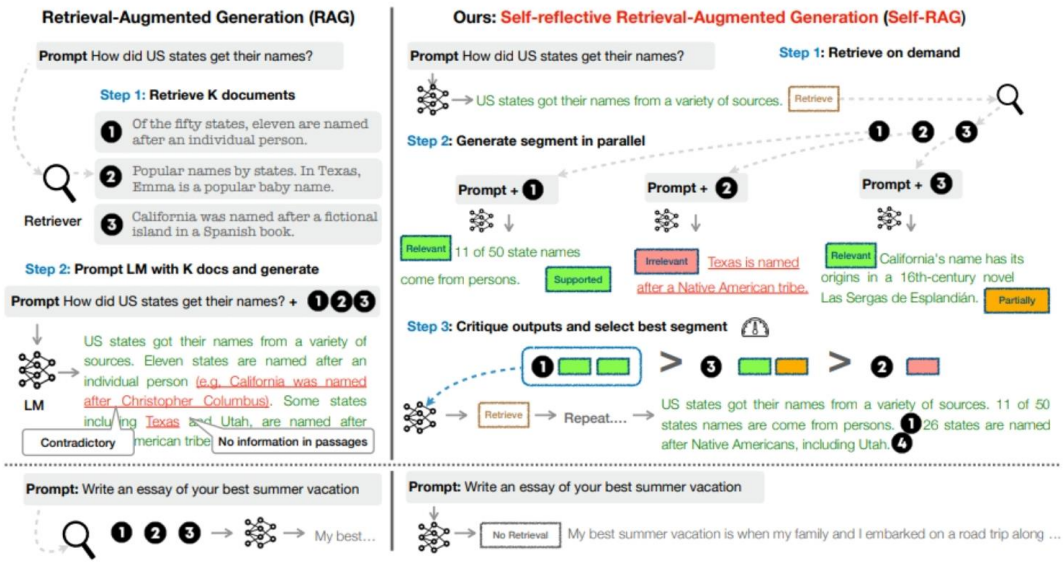


Figure 1: Overview of SELF-RAG. SELF-RAG learns to retrieve, critique, and generate text passages to enhance overall generation quality, factuality, and verifiability.

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？

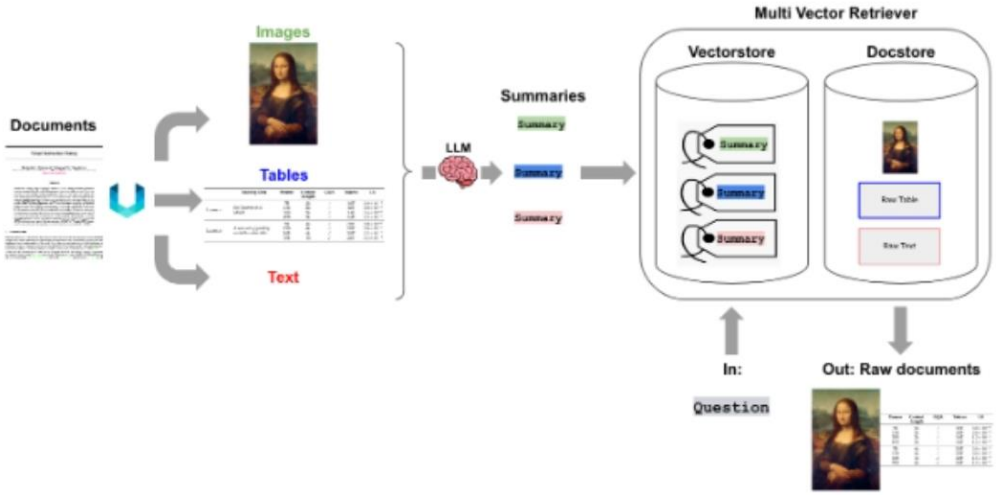
Self-RAG 使用反思字符来自我评估输出，使其在推理过程中具有适应性。 根据任务的不同，可以定制模型，通过检索更多段落来优先考虑事实准确性，或强调开放式任务的创造力。 该

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：

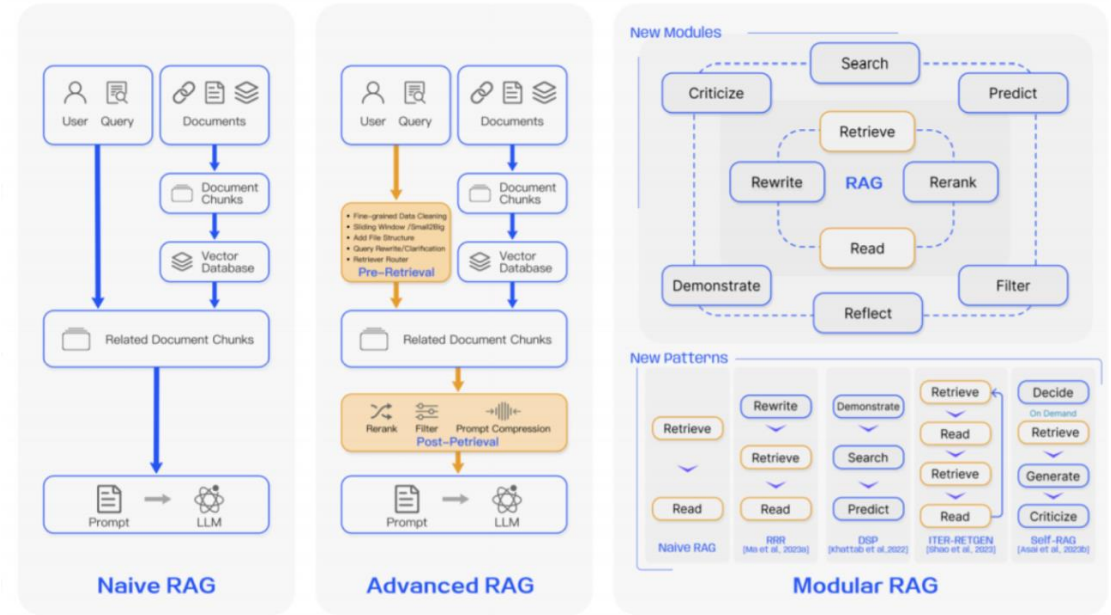
增加了相关文本块的召回率；

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



动机：打破了传统的“原始 RAG”框架，这个框架原本涉及索引、检索和生成，现在提供了更广泛的多样性和更高的灵活性。
模块介绍：

搜索模块： 融合了直接在（附加的）语料库中进行搜索的方法。这些方法包括利用大语言模型（LLM）生成的代码、SQL、Cypher 等查询语言，或是其他定制工具。其搜索数据源多样，涵盖搜索引擎、文本数据、表格数据或知识图等。

记忆模块： 本模块充分利用大语言模型本身的记忆功能来引导信息检索。其核心原则是寻找与当前输入最为匹配的记忆。这种增强检索的生成模型能够利用其自身的输出来自我提升，在推理过程中使文本更加贴近数据分布，而非仅依赖训练数据。

额外生成模块： 面对检索内容中的冗余和噪声问题，这个模块通过大语言模型生成必要的上下文，而非直接从数据源进行检索。通过这种方式，由大语言模型生成的内容更可能包含与检索任务相关的信息。

任务适应模块： 该模块致力于将 RAG 调整以适应各种下游任务。

对齐模块： 在 RAG 的应用中，查询与文本之间的对齐一直是影响效果的关键因素。在模块化 RAG 的发展中，研究者们发现，在检索器中添加一个可训练的 Adapter 模块能有效解决对齐问题。

验证模块： 在现实世界中，我们无法总是保证检索到的信息的可靠性。检索到不相关的数据可能会导致大语言模型产生错误信息。因此，可以在检索文档后加入一个额外的验证模块，以评估检索到的文档与查询之间的相关性，这样做可以提升 RAG 的鲁棒性。

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。

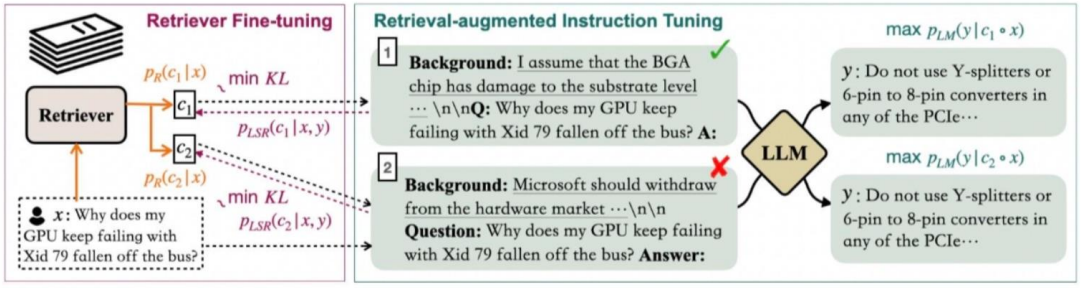
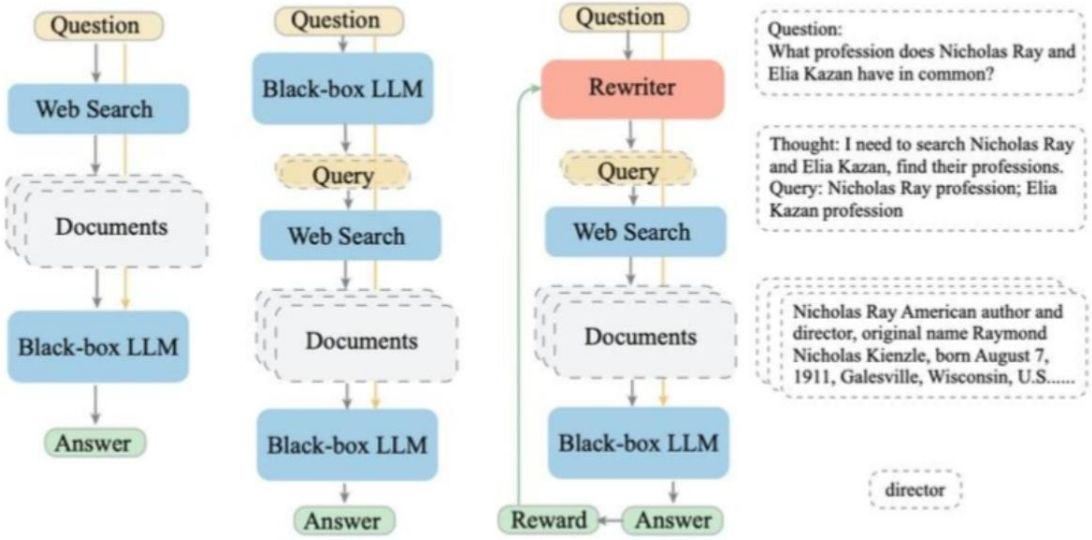


Figure 1: The RA-DIT approach separately fine-tunes the LLM and the retriever. For a given example, the LM-ft component updates the LLM to maximize the likelihood of the correct answer given the retrieval-augmented instructions (§2.3); the R-ft component updates the retriever to minimize the KL-Divergence 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，Flowise AI6 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 也在不断发展。

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。

大模型（LLMs）增量预训练篇

大模型（LLMs）Tokenizer 篇

大模型（LLMs）显存问题篇

大模型（LLMs）分布式训练篇

大模型（LLMs）加速篇

大模型（LLMs）蒸馏篇