自然语言处理
Natural Language Processing

第8章　人工智能大模型

授课教师：　黄河燕

授课时间：2024.11

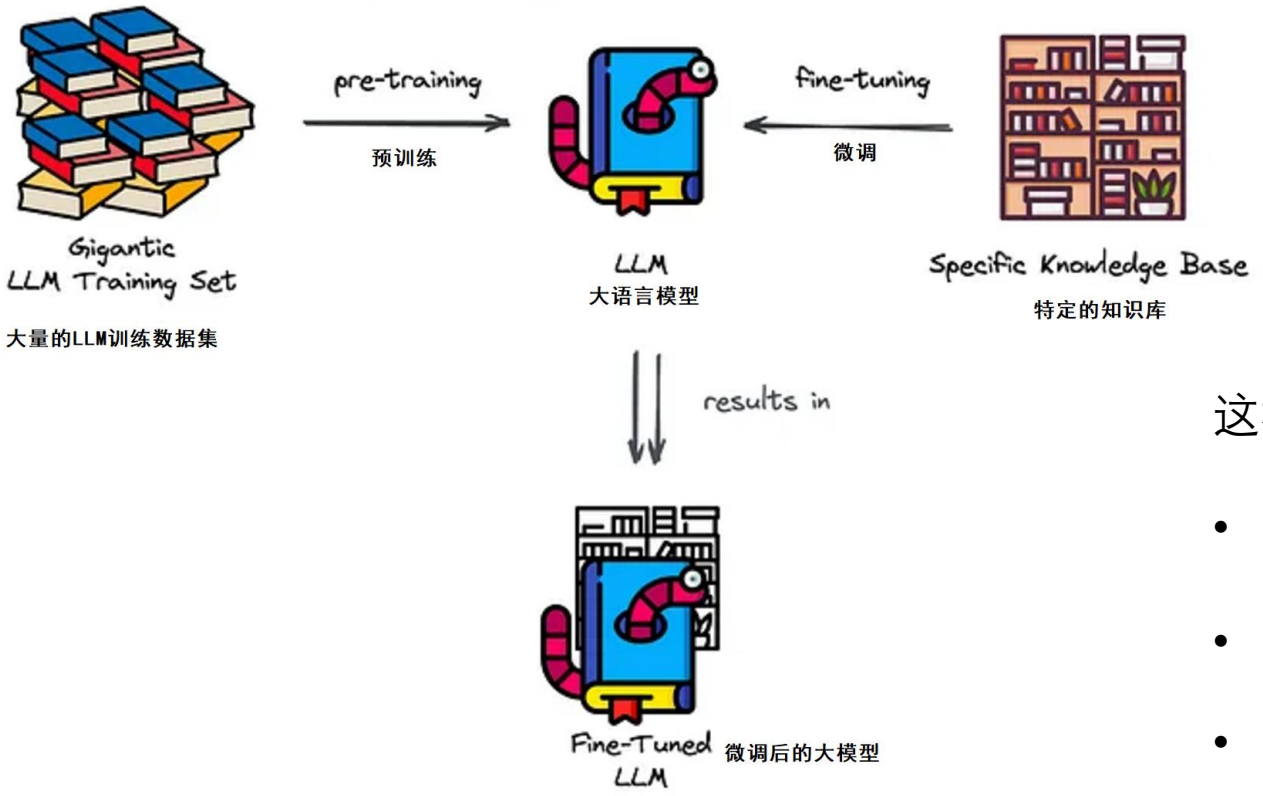
8.5 检索式生成 (RAG)

1. 引言
2. 检索器 Retriever
3. 检索式生成的技术与方法
4. 现状和发展趋势
5. 总结

第一节

引言

大模型更新参数知识



这样做有什么缺点呢?

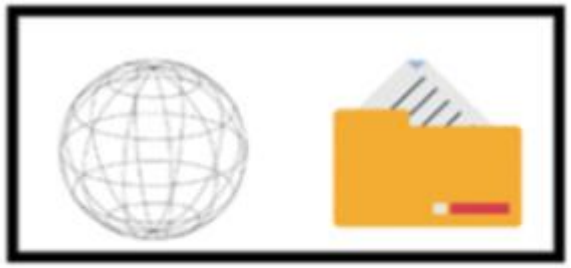
- 需要专业的人员来进行再训练（微调）
- 需要高质量标注数据来保证模型的训练效果
- 需要大量的训练资源（比如：GPU，电力）。
- 大模型没法分辨哪些是“通用知识”和“专有知识”，无法应对需要精准溯源的场景。

大模型知识的分类

1、参数化知识（Parametric-Memory）：

LLM

2.非参数化的知识（Non-parametric-Memory）：



Non-parametric Memory

大模型知识的分类

1、参数化知识（Parametric-Memory）：

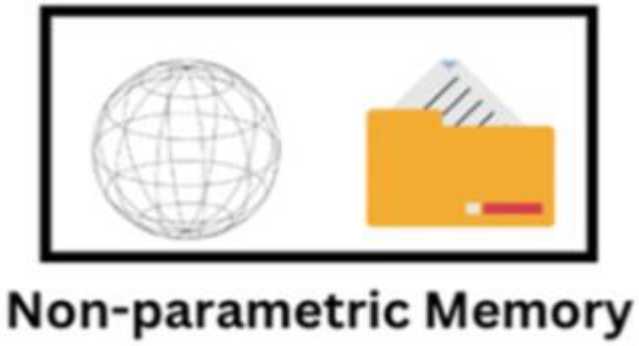
优点：

- 知识利用效率高
- 无需接入外部知识库，也有可观的性能

缺点：

- 容易产生幻觉，对知识理解产生偏差。
- 训练成本高昂
- 不容易溯源

2.非参数化的知识（Non-parametric-Memory）：



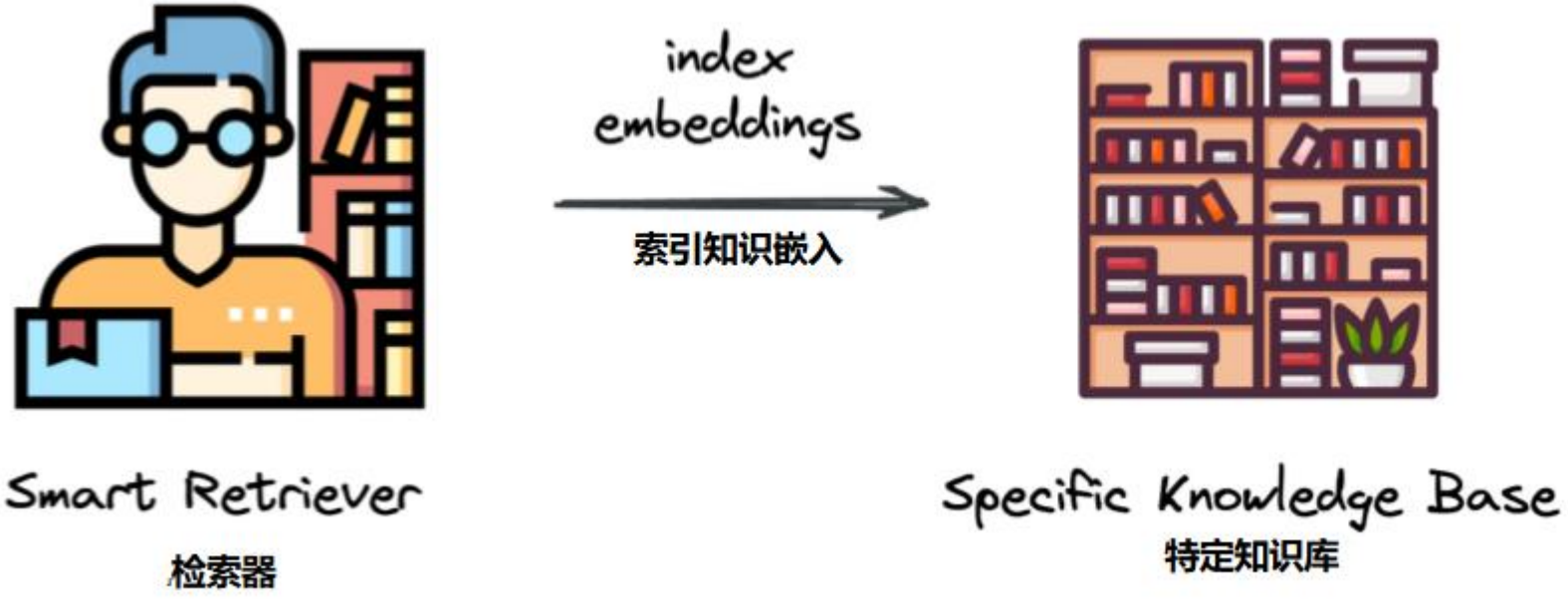
优点：

- 知识有较高的精准度
- 可以精准溯源

缺点：

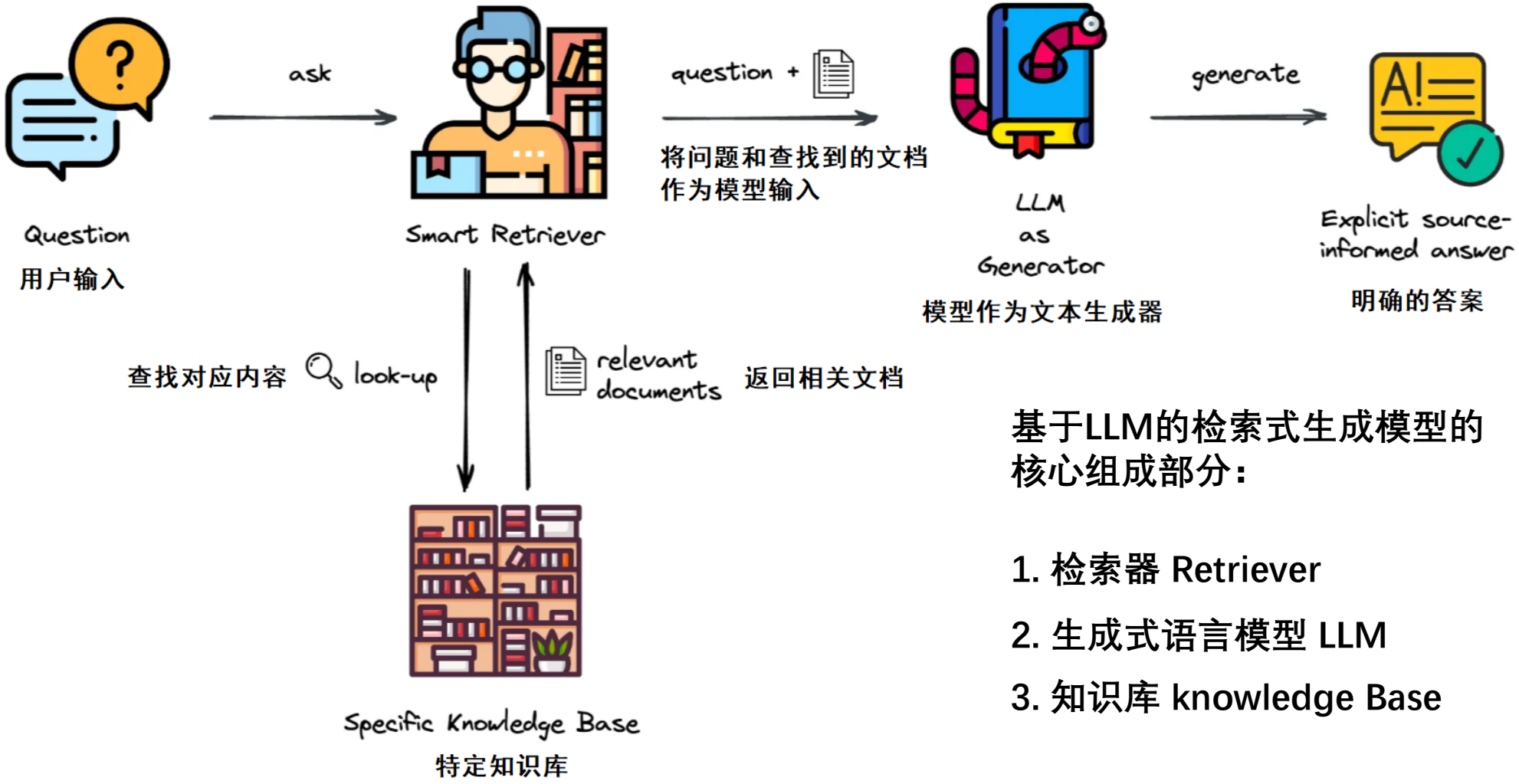
- 需要一定的存储空间
- 原生数据没有做适应大模型的分类，需要搭配检索器使用

检索式生成 (RAG) Retrieval-Augmented-Generation



这描述了一个语义搜索引擎。在这种情况下，嵌入(Embedding) 是文档部分的矢量表示(Vectorial Representations)，它们允许对每个部分中存储的实际含义进行编码。通过比较嵌入，我们可以确定哪些文本部分在意义上与其他文本部分相似。

基于大模型的检索式生成

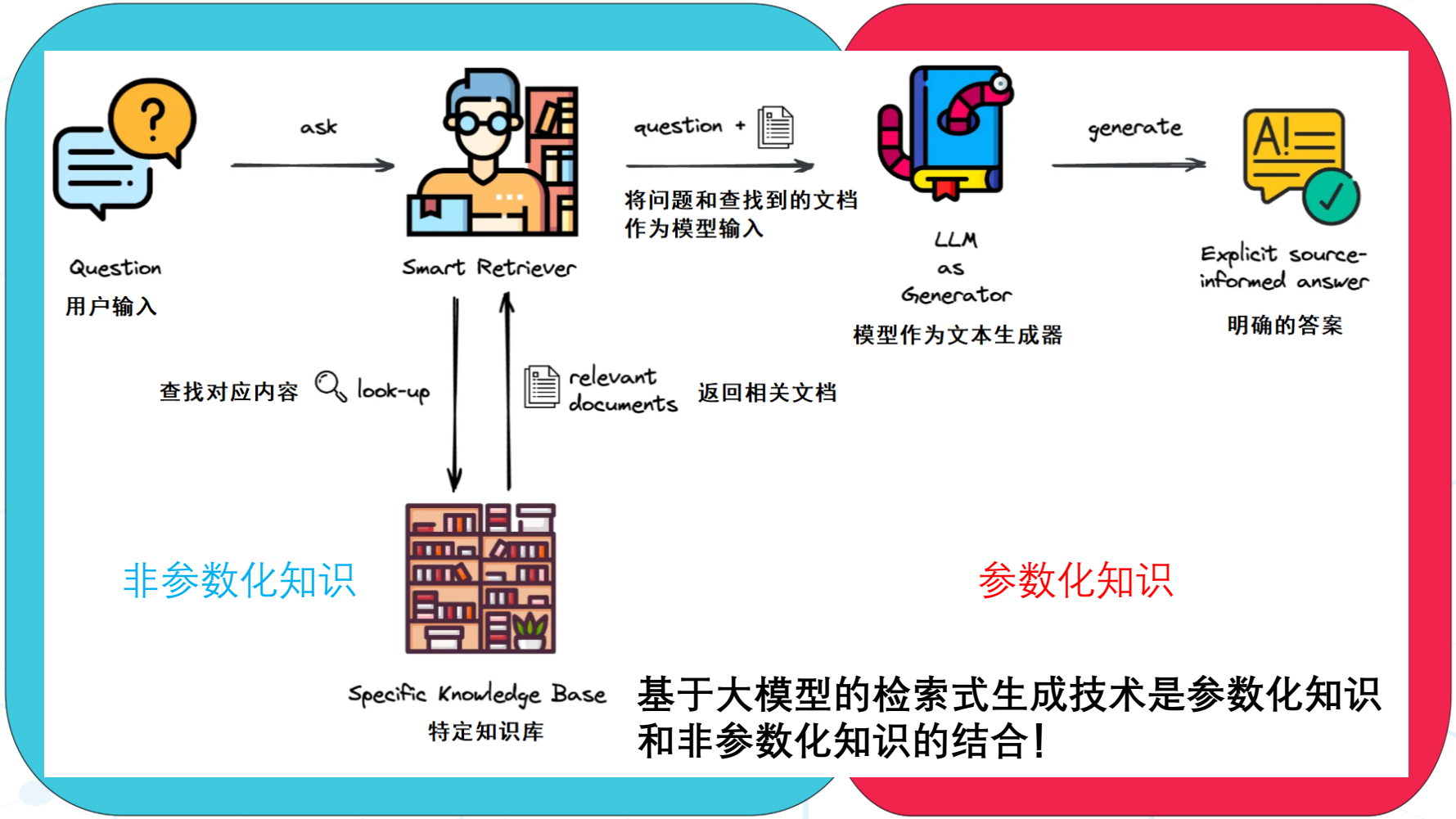


基于LLM的检索式生成模型的核心组成部分：

1. 检索器 Retriever
2. 生成式语言模型 LLM
3. 知识库 knowledge Base

基于大模型的检索式生成

提问：在下图所示的结构图中，哪一部分属于利用利用非参数化知识，哪一部分属于利用参数化知识？



第二节

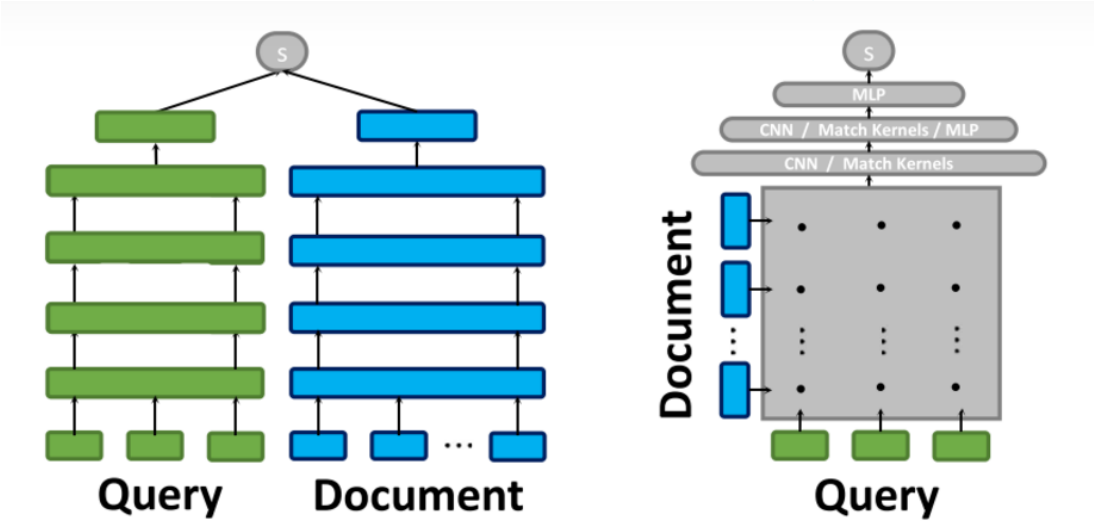
检索器 Retriever

检索器的类型

目前检索式生成（RAG）中使用的主流检索器可以主要被分为两类：

1.基于统计模型的检索器：代表模型 **BM25**

2.基于深度学习模型的检索器： 代表模型 **DPR**

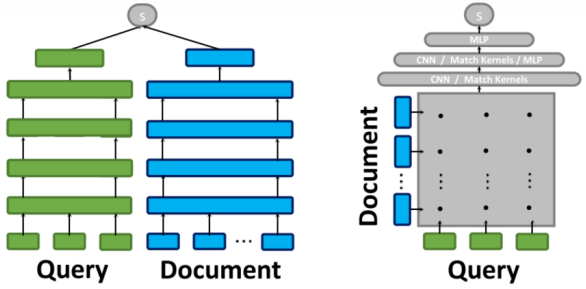


检索器的类型

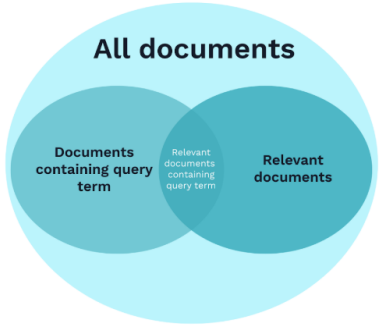
目前检索式生成（RAG）中使用的主流检索器可以主要被分为两类：

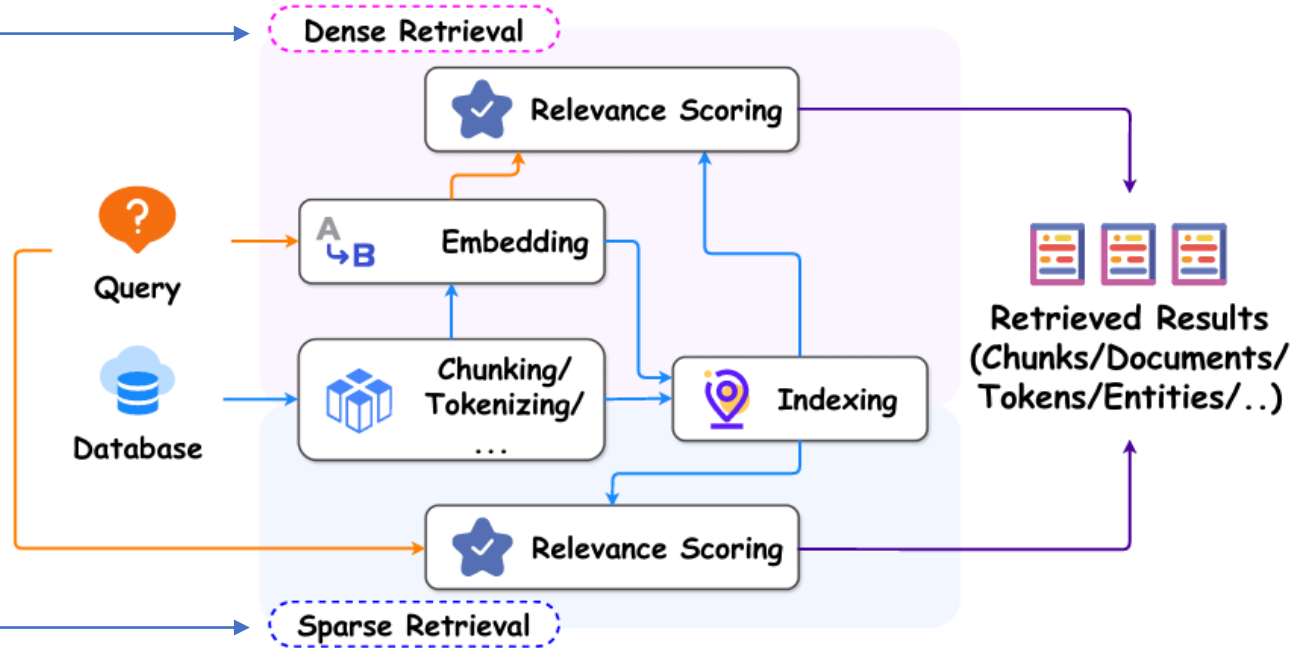
提问：这张图包含两类检索器，哪一个可以代表DPR算法，哪一个可以代表BM25算法呢？

DPR （Dense-Passage-Retrieval）



Best Match 25





检索器 Retriever - BM25

1、基于统计模型算法的检索器：**BM25 （Best Matching）是当前最主流的文本匹配算法，BM25可以视作TF-IDF算法的优化，主要内容是计算query到文档集合的相似度得分**

$$tf-idf_{score} = tf \times idf = \frac{\text{某文档中目标词出现的数量}}{\text{某文档总词数}} \times log \frac{\text{文档总数}}{\text{包含目标词的文档数量}}$$

TF-IDF 算法： 定义：TF = Term Frequency (词频) IDF = Inverse Document Frequency （逆文档频率）

TF-IDF倾向于过滤掉常见的词语，保留重要的词语， 当一个词出现频率越高说明这个词重要性低（比如：am, is ,are）

BM25算法：

- BM25在TF-IDF的基础上增加了几个可调节的参数，使其在应用中更具灵活性和实用性。
- BM25对于词频、逆文档频率以及字段长度的归一化具有更合理的定义。
- 在词频的重要性方面，BM25有一个上限，即随着词频增长，词的重要性增长程度会被限制。

BM25的公式

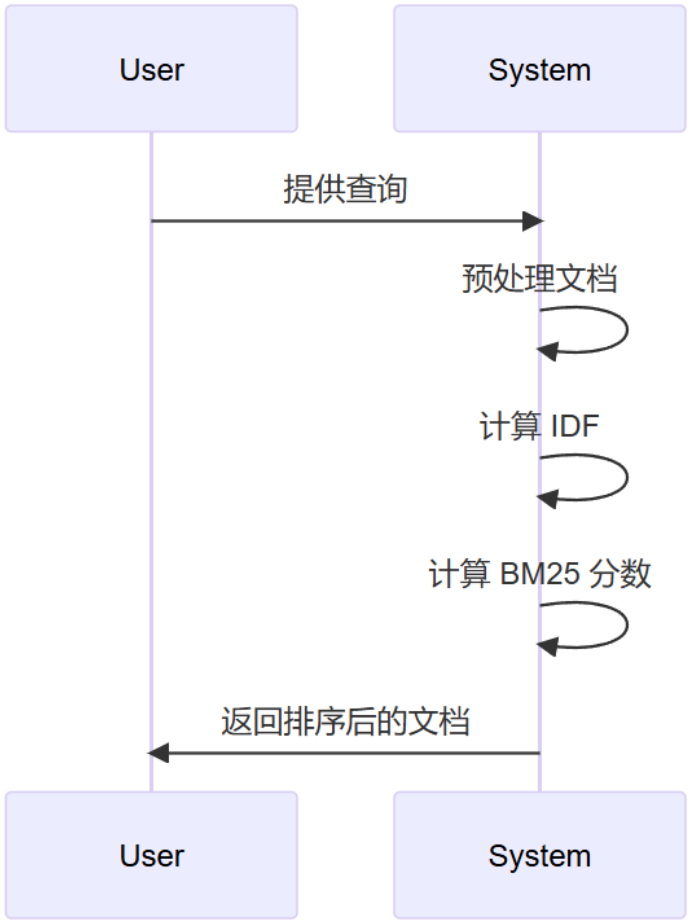
$$\text{score}(Q,d) = \sum_{i=1}^{n} w_i R(q_i,d)$$

其中Wi 的计算方式同IDF类似：

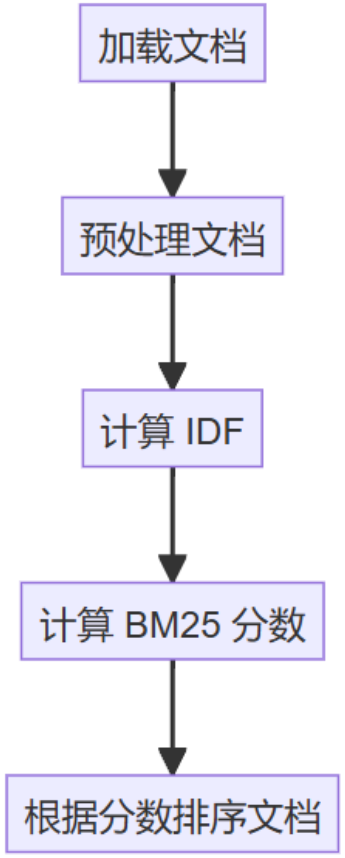
$$w_i = idf_{q_i} = log \frac{N - df_i + 0.5}{df_i + 0.5}$$

检索器 Retriever - BM25

BM25检索器的流程图：



BM25算法的处理序列：



检索器 Retriever - DPR

2、**DPR 全称为 Dense-Passage-Retrieval**，是最早提出使用嵌入向量来实现文档检索的模型，是一种用于开放领域问答（Open-Domain-Question-Anwsering，ODQA）任务的检索方法。它的核心思想是利用深度学习模型来生成问题的高维密集向量表示，并在大量的文档集合（例如：维基百科）中检索与问题最相关的文档段落。

DPR的架构基于以下两个主要组件:

- 查询编码器（Query Encoder）：基于BERT-base模型的编码器，用于将输入换成一个高维的查询向量。
- 文档编码器（Document Encoder）：同样基于BERT-base模型，该编码器将文档中的每个段落转换成一个高维的文档向量。

DPR使用一种称为最大内积搜索(Maximum Inner Product Search，MIPS)的方法来找到与查询向量最相关的文档向量。这种方法可以高效地在大规模数据集中检索最相似的项，通常用于高维空间中的相似性搜索。

检索器 Retriever - DPR

问题形式: 我们有一堆文档 D ={d1,d2,…….,dn}，将这里面的每个文档切分为多个等长的 passages, passage 就是检索结果的基本单元。这些切分后的 passages 构成了我们的语料库 C = 〔p1,p2,…,Pm}。

而我们的任务是，给定一个问题 question q，我们需要返回与其相关的 passage 集合 CF ∈C。如之前所说，DPR 是双编码器架构（dual-encoder）。

于是我们用公式化语言定义一下编码器:

文档编码器 Ep(·): 利用BERT模型，将任意的 passage 映射为 d维的 embedding 向量·
问题编码器 Eq(·):利用BERT模型，将一个 question 映射为d维的 embedding 向量

当我们得到文档和问题的嵌入（embeddings）之后，计算两个嵌入相似性使用的是向量点积（dot product）：

$$\mathrm{sim}(q,p) = E_Q(q)^\intercal E_P(p).$$

检索器 Retriever - DPR

Dense Retriever 简单举例：

假设我们有两个文档经过BERT模型编码后为以下表式：

1. [0.13，0.11，0.52，0.55]

2. [0.25，0.35，0.38，0.65]

我们的用户问题输入经过编码后是：[0.10，0.10，0.0， 0.20]

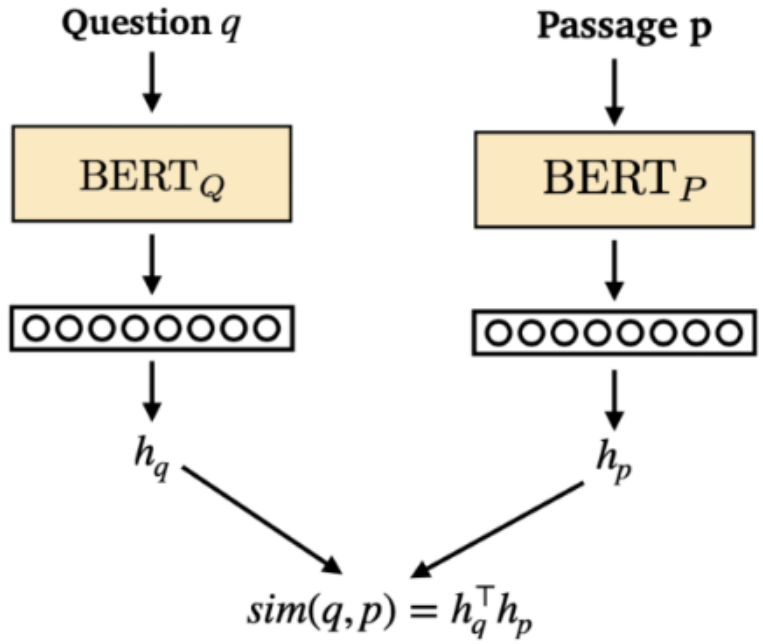
接下来算算看，哪个文档和输入的相关度最高？（提示：点积计算）

答案：
第一个文档得分是：
[0.13*0.1，0.11*0.1，0.52*0，0.55*0.2] = **0.013+0.011+0.11=0.134**

第二个文档得分是：
[0.25*0.1，0.35* 0.10 ，0.38，0.65] = **0.025+0.035+0.13=0.19**

通过对比，第二个文档得分最高，说明第二个文档最相关。

DPR图示：



检索器总结

我们介绍了两种主流的检索器算法，提问：那么他们的优缺点和应用方向是什么呢？

1. 基于统计数学模型的BM25算法：

优点：

1. 模型基于数学公式计算来排序，不需要训练
2. 部署简单，对算力要求不高，能够高效运行
3. 算法复杂度较低,能在较短的时间内处理大量数据
4. 适合关键词匹配的精准提取

缺点：

1. 对于复杂的问题，检索效果不够好，因为BM25没有构建文档段落之间的关系，缺乏语义级别的理解。
2. 对于单一文档，没有上下文的关系建模
3. 假设文档中没有出现问题中的关键词，会出现无法检索到相关问题答案。（可以打开部分手机app试试，便可以猜出自己使用的哪些APP使用了BM25算法）

应用方向：

搜索引擎

推荐系统

检索器总结

我们介绍了两种主流的检索器算法，提问：那么他们的优缺点和应用方向是什么呢？

1. 基于深度学习模型的DPR算法：

优点：

1. 可以在没有问题关键词的文档中，进行语义匹配
2. 匹配语句时，考虑了上下文关系，更加精确
3. 利用最大内积搜索(MIPS)，实现了高效检索
4. 更高效的后期交互和细粒度相似度评估，进一步提升了检索的准确性和相关性

缺点：

1. 模型受编码器基座模型的性能影响，例如BERT
2. 模型需要训练，检索器效果上限由训练效果决定
3. 内存占用量大，需要搭配向量压缩算法来实现工业落地

应用方向：

大模型问答系统

文心一言

通义千问

AI搜索

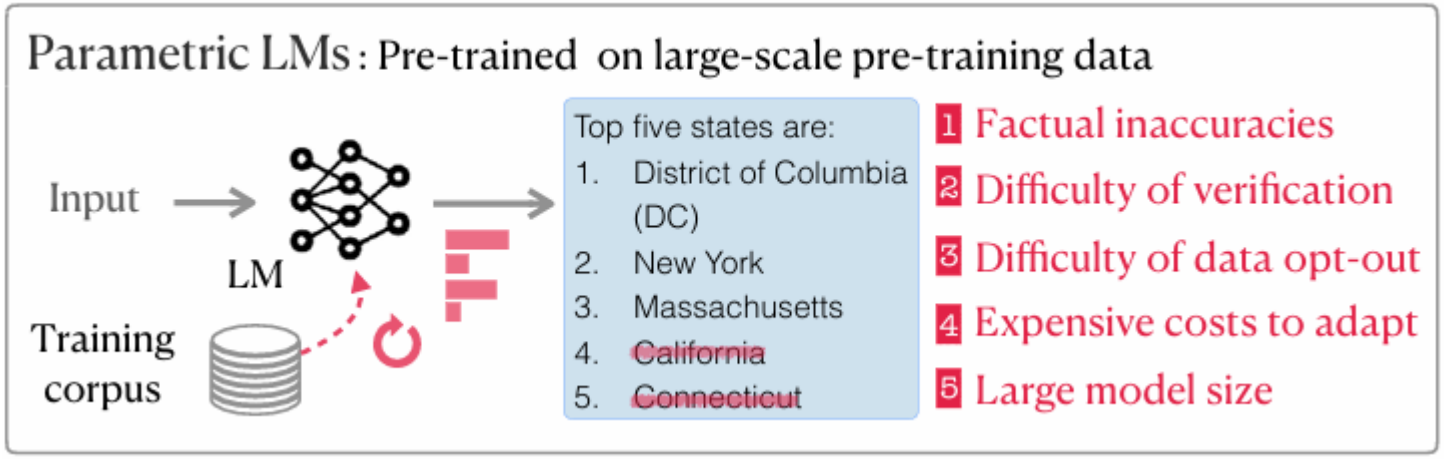
Microsoft Bing

Apple Intelligence

第三节

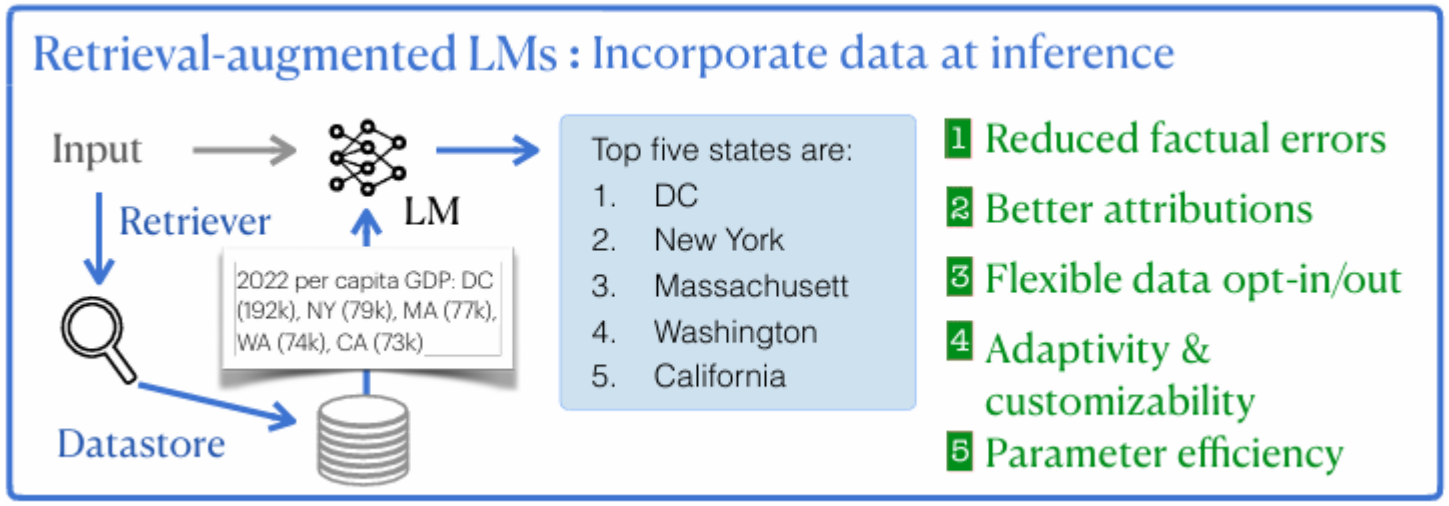
检索式生成(RAG)的技术与方法

检索式生成与大语言模型



大语言模型存在的问题：

1. 对事实把握不精确
2. 难以对含有事实的输出进行查证（溯源）
3. 难以控制信息输出（例如：隐私，版权）
4. 知识更新的成本过高
5. 模型的参数过多（训练，推理成本也更高）



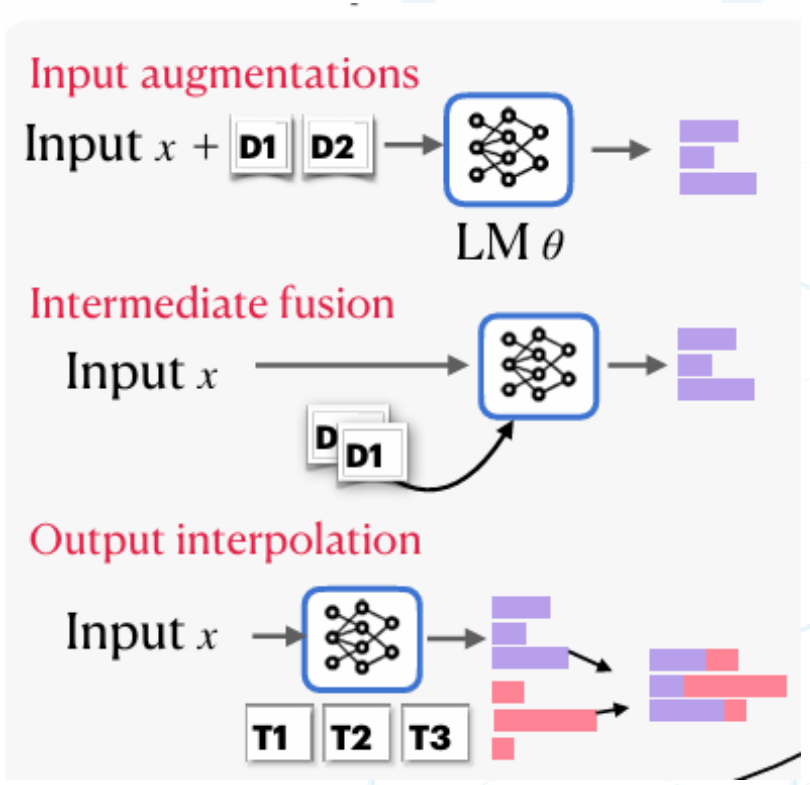
检索式生成大语言模型：

1. 减少输出中的事实性错误
2. 可以对精准溯源
3. 信息的输出可控
4. 具有很强的可塑性&拓展性
5. 参数的高效利用

检索式生成(RAG)的技术与方法

检索式生成语言模型的方法架构大致可以分为三种类型：

1. **输入增强 （input augmentation）**
2. **中间融合 （Intermediate fusion）**
3. **插值输出（output interpolation ）**

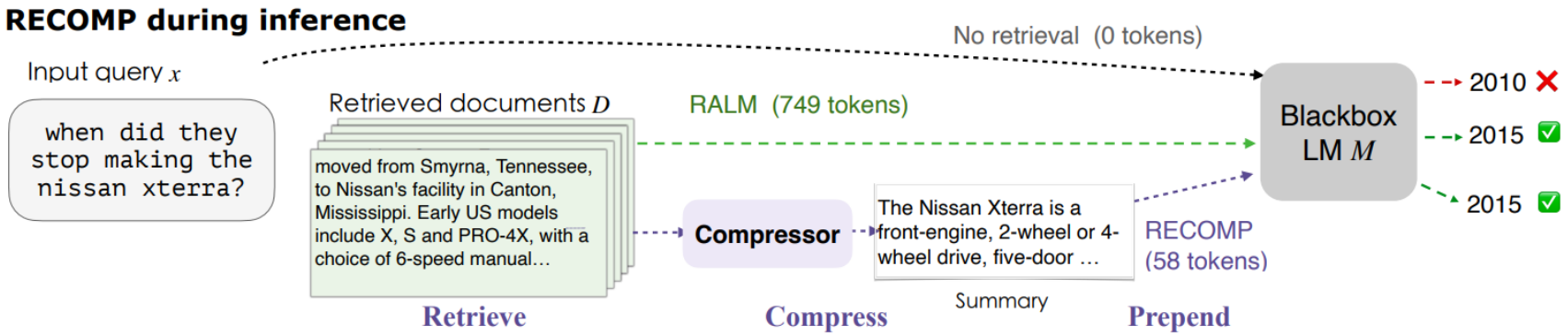


输入增强和**中期融合**通常涉及检索文本块，并使用语言模型对检索到的文本块进行处理。 而**输出插值**直接检索连续的标记或短语，这种方法会产生更大的索引。

检索式生成(RAG)的技术与方法

1. 输入增强 （input augmentation）：输入增强的通常架构都是将检索到文本集成到大模型的上下文中从而实现检索式生成，这种方法简单直接，易于实现。

- 输入增强用输入空间中的检索结果 z 来增强原始输入 x， 然后运行标准的语言模型来进行推理。
- 输入增强使得不同模型的灵活插件能够用于检索和语言模型

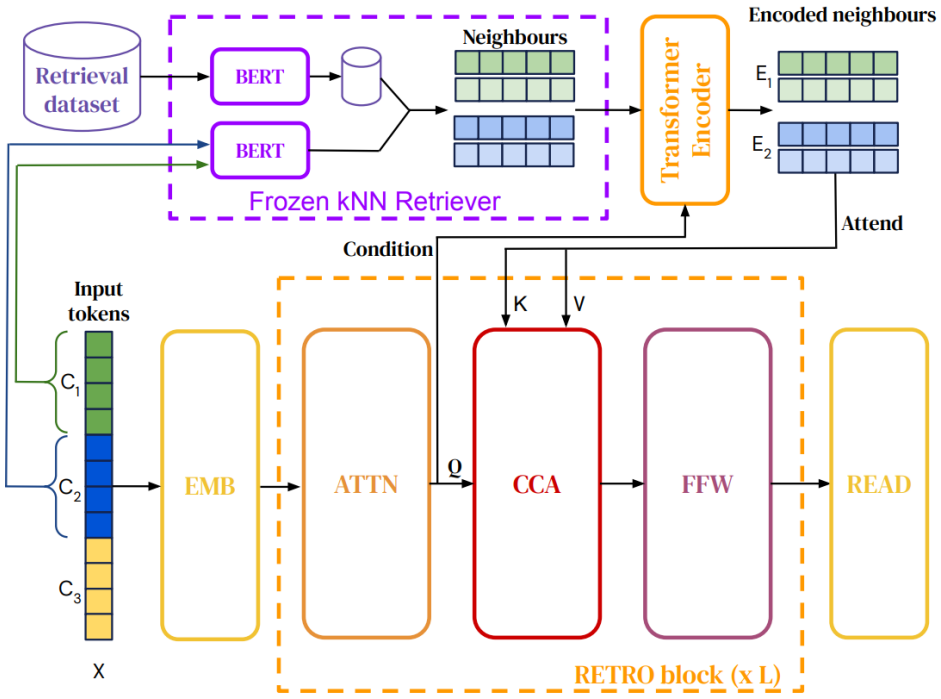


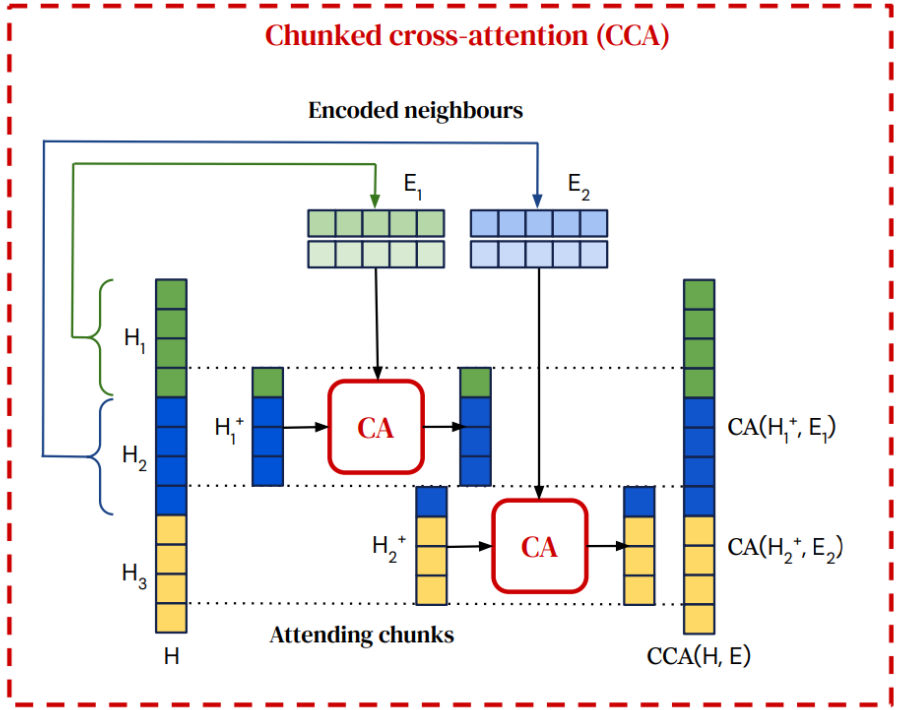
RECOMP (Xu et al., 2024).是发表在 ICLR的一篇文章，文章引入了一个压缩器，在对外部数据经过了检索与重排序阶段后，可以从知识库中获得与查询相关的若干文档，然而文档级别的知识具有较多的 token 量，增大了推理的开销。因此可以在将相关文档集成到上下文之前进行压缩，再输入到大模型中。这是一个典型的**输入增强**的例子。

检索式生成(RAG)的技术与方法

2. 中间融合（ Intermediate fusion ）：

- 为了以更可扩展的方式整合检索到的结果，RETRO（Borgeaud等人，2022）引入了一种新的注意力机制，该机制接收许多独立于查询 x 的预编码文本块，并同时将它们合并到中间空间中。
- RETRO++（Wang et al，2023b）和 InstructRetro（Wang et al，2023a）在仅包含解码器的语言模型上 (类 GPT)，展示了该方法的有效性。





检索式生成(RAG)的技术与方法

3. 插值输出（Output interpolation）：插值输出是在语言模型的token分布中插入检索到的文本token，来实现对语言模型的外部知识注入。

- 第一种方式是利用 **kNN-LM**（Khandelwal et al，2020）的方法，使用检索到的token分布插值语言模型的token分布，而不需要额外的训练
- 第二种方法是通过设置新的训练目标，或在数据存储中的每个短语上用非参数分布完全替换参数分布来扩展这一方向

插值输出的有关论文:

Generalization through Memorization: Nearest Neighbor Language Models (Urvashi Khandelwal, ICLR 2020)

Training Language Models with Memory Augmentation (Zhong et al., EMNLP 2022)

Nonparametric masked language modeling. （Min S et al., ACL 2023 Findings）

Copy is all you need (T Lan et al., ICLR 2023)

检索式生成(RAG)的技术与方法

总结：

1. 输入增强 （input augmentation）：

输入增强方法应用起来简单，可以直接和语言模型结合。大多数直接使用现成大模型的检索式生成方法都属于这一类。值得注意的是，这类方法的缺点也很明显，那就是在大模型的prompt中输入过多的文本段落信息会加大模型资源的消耗，其次大模型需要具有Long Context能力，也就是长文本能力来适配这一方法。

2. 中间融合 （Intermediate fusion）：

中间融合方法需要对模型架构进行大刀阔斧的改造，同时对于引入新的预编码文本块也需要额外的预训练。

3. 插值输出（output interpolation ）：

差值输出方法相比于前两个方法，需要构造更大的索引。因为这种方法需要对数据库中所有的嵌入（embeddings）进行编码

哪一种方法是现在工业界最主流的方法呢?

答：输入增强，因为可以直接与类GPT的大语言模型搭配，且无需额外对大模型结构的改造。

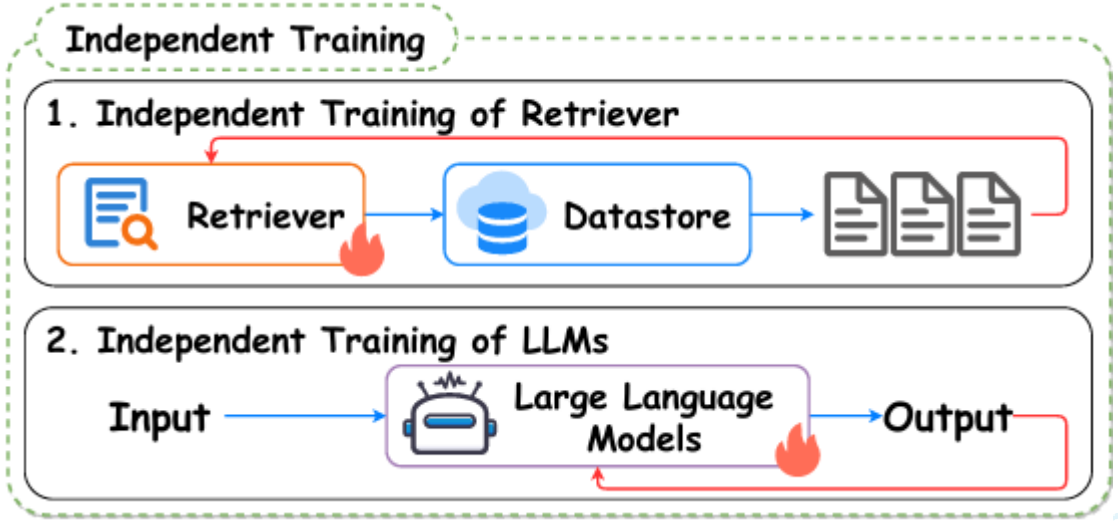
检索式生成(RAG)的技术与方法

检索式生成由三个主要部分组成：索引、检索器（即生成输入和文档编码的模型）和语言模型。如何高效地同时更新它们以优化整个流程仍然是一个具有挑战性的问题。

检索式生成语言模型的训练一般分为三种：

1. 独立训练（Independent Training）：

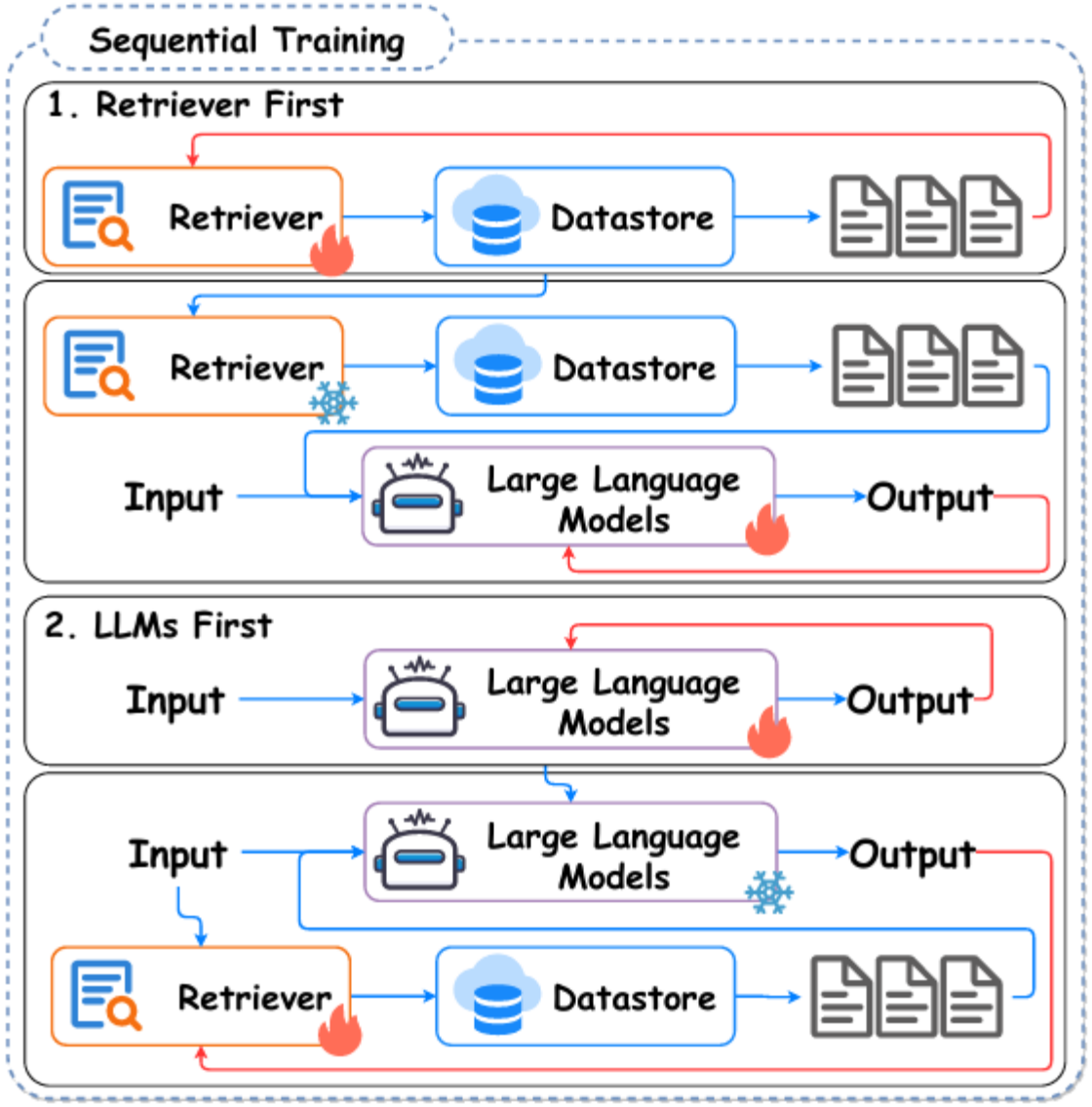
独立训练指的是在方法中，只对检索器进行训练，或者只对语言模型训练，且两个组件之间在训练过程中没有互动关系。



检索式生成(RAG)的技术与方法

2. 序列训练（Sequential Training）：

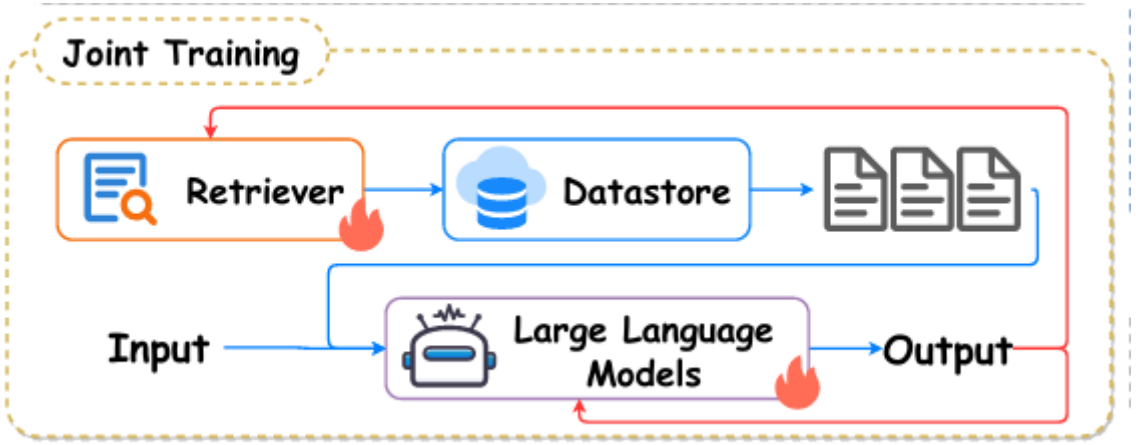
如果我们希望既对检索器进行训练，使其更好的匹配相关的数据库，又希望语言模型通过学习来熟悉数据库的数据来实现效果更好的检索式生成。那么，序列训练是方法之一，我们分为两步进行训练，训练第一步的时候，只训练其中的一个模块，等这个模块训练完毕以后，冻住其中的参数，再训练另外一个模块。



检索式生成(RAG)的技术与方法

3. 联合训练 （Joint Training）：

跟序列训练的目的一样，联合训练同时对检索器和语言模型进行了训练，但不同的是，联合训练没有将训练步骤分开，它同时训练检索器和语言模型。这种方法增强了检索模型和语言模型之间的信息交互，是性能最优的方法，但是也是训练难度和资源开支最大的方法。



检索式生成(RAG)的技术与方法

训练方法总结：

- **单一训练和序列训练（Independent or sequential training）：**

单一训练和序列训练指的是把检索器和语言模型分开训练，即两个组件之间在训练过程中没有互动关系。

- **联合训练** （Joint Training）：

联合训练同时训练语言模型和检索组件，以进一步优化它们的交互和端到端检索增强的语言模型。

联合训练中的一个不可忽视的挑战是，在训练过程中更新检索器模型和结果索引会产生大量的计算开销。在每个时间步中，为数据存储中的数百万或数十亿个文档重复生成嵌入是不切实际的。

尽管**单一训练和序列训练**缺少模型和检索器之间的互动，同时对于性能而言也是次优解，但是他的优点是易于部署，消耗资源少，训练难度低，所以是工业界的常用首选训练方式。

联合训练通常拥有最好的性能，但是它的训练成本过高，虽然在工业界不常用，但是近期学术领域正在研究只让模型一次只根据一个小的分索引来学习一部分检索文本，来逐渐建立对所有文本的索引，而不是让模型去一次性学习全部的检索文本。这种方法看似提高了效率，但仍需要更多的探索。

检索式生成(RAG)的技术与方法

除了三种涉及到训练的检索式生成的方法，还有一种完全不需要训练的方法。这种方法又被称作 **In-Context learning**方法，又被叫做**上下文学习**。

In-context Learning 上下文学习是一种学习范式，是现代自然语言处理领域中一种重要的学习方法，尤其在使用大规模，尤其在使用大规模预训练模型时，它允许模型在给定的上下文中进行学习和推理，而无需真正更新模型参数。

1. In-context Learning （ICL）上下文学习 的核心概念：

- **上下文依赖：**ICL的核心在于利用模型的上下文理解能力来完成任务，模型根据输入的上下文信息(包括示例和任务描述)进行推理，而不是依赖于显示的训练过程。

- **无参数更新**：ICL不涉及对模型实际参数的修改，模型保持预训练状态，只是根据提供的上下文信息调整其生成或分类行为。

- **动态适应**：模型在推理时会动态地适应给定的上下文，通过分析上下文中的示例或指示来生成合适的输出，这种适应能力来源于模型在预训练阶段学到的通用知识。

检索式生成(RAG)的技术与方法

回顾一下提示词学习的工作原理 （我们上节课讲过）：

提示词和示例：

- **提示词：**ICL常通过提示词来引导模型的生成过程，提示词通常包括任务描述，问题陈述或请求模型执行的操作。
- **示例：**在少样本学习(Few-Shot Learning)中，提示词可能包括一些示例输入和输出，帮助模型理解如何处理类似的任务

上下文提供：

- **任务描述**：在ICL中，任务描述用于告诉模型要完成的任务，例如：生成一个关于人工智能的总结。
- **示例输入输出**：提供几个示例输入和输出对，可以帮助模型理解特定任务的模式或要求，例如：给出一些翻译示例来帮助模型进行语言翻译。

检索式生成(RAG)的技术与方法

通过上下文学习来实现检索式生成语言模型也是检索式生成的主流方法之一：

TACL收录了一篇In-Context Retrieval-Augmented Language Models 简称（IC-RALM）（Ori Ram, et al, TACL）就是通过上下文学习来实现的检索式生成。

这种方法既不需要对检索器训练，也不需要对语言模型进行训练，直接通过大模型的上下文学习能力，来对检索到的内容进行学习，从而输出相应的答案。

第四节

现状和发展趋势

检索式生成的现状和发展趋势

1、检索器和语言模型的局限性：

- 尽管检索增强型语言模型在知识密集型任务中取得了成功，但是他们在广泛的应用中扔有局限性。例如，检索增强的语言模型在推理任务上只会产生很小的边际收益，这可能是由于检索和大模型语言的缺点造成的。

2、检索器和语言模型缺乏互动：

- 如同之前所说，常见的方法，通常直接将检索到的结果附加到预训练语言模型的输入中，并采用输入增强（input augmentation）。然而，这些方法在整个训练和推理过程中缺乏检索和语言模型组件之间的密切交互。

- 输入增强这种方法，由于检索器并非完美，会导致不相关的文本可能会被检索到且添加到语言模型的输入中，这样对于文本生成会造成干扰，降低文本的生成质量。

- 如果是基于多文档的输入，大语言模型特性会导致长文本能力不强的语言模型忽视很大一部分输入，从而产生输出内容的不稳定。

检索式生成的现状和发展趋势

3、缺乏对于检索式生成模型的基础架构开发：

- 例如，PyTorch FSDP或DeepSpeed等开源软件通过全分片数据并行(FSDP)等技术实现了资源高效的大语言模型预训练。相对于大语言模型，从方法论和基础设施的角度来看，检索增强语言模型训练过程的优化研究相对较少。虽然上述开源软件可以用来对检索式生成的语言模块进行改进，但仍然缺乏的是解决检索式生成特有的一些问题。

- 在训练过程中同步更新大规模索引会引入大量的计算开销，如何在正常计算环境下有效地更新索引仍然具有挑战性

- 检索式生成会比常规的大预言模型需要更多的推理计算资源，特别是在庞大的数据库场景下（例如：超过 1 万亿的token的数据）。

检索式生成的现状和发展趋势

重新思考如何发展更好的生成式检索技术?

1. 发展更好的检索器（Retriever）：

- 结合之前的所讲的两类检索器，一类是基于关键词匹配（BM25），另一类是基于语义匹配（DPR）。然而我们需要更强大的检索器不限于关键词匹配和语义匹配，才能使检索器的性能更加强大且鲁棒。

2. 改进数据库（Datastore）：

- 目前针对数据库的改进相对而言稀缺，我们需要思考如果构建更好的数据库。

  例如：如何对数据库进行质量过滤？如何平衡一个数据库中不同领域的数据量？

检索式生成的现状和发展趋势

重新思考如何发展更好的生成式检索技术?

3. 增强检索器和语言模型之间的互通:

- 研发不同于输入增强的新模型架构，例如，最新的一篇论文(Muennighoff, 2024)提出了GRIT (Generative representational instruction tuning)架构，来训练一个模型同时应对检索和生成任务。这个方法通过缓存文本内容表示，减少了检索式生成的延迟。

4. 让检索参与到大语言模型的预训练中:

- 之前我们讨论过单一训练和序列训练的语言模型的训练中，检索器没有参与到语言模型训练的部分。我们需要研发一些新的方法，在不改变语言模型的架构下，让检索器对整个检索系统进行额外的指导，从而使检索式生成的整体性能更加优秀。

检索式生成的现状和发展趋势

重新思考如何发展更好的生成式检索技术?

5. 预训练后的进一步适配:

- 大刀阔斧的架构修改或预训练是需要大量计算资源。另一个方向是探索预训练后检索式生成语言模型的再适应(adaptation)。例如：在检索式生成中使用instruction following和RLHF技术。这些技术可以保证检索式生成语言模型能够输出更符合人类要求的回答（比如：回答内容偏好，回答方式，语气等）。

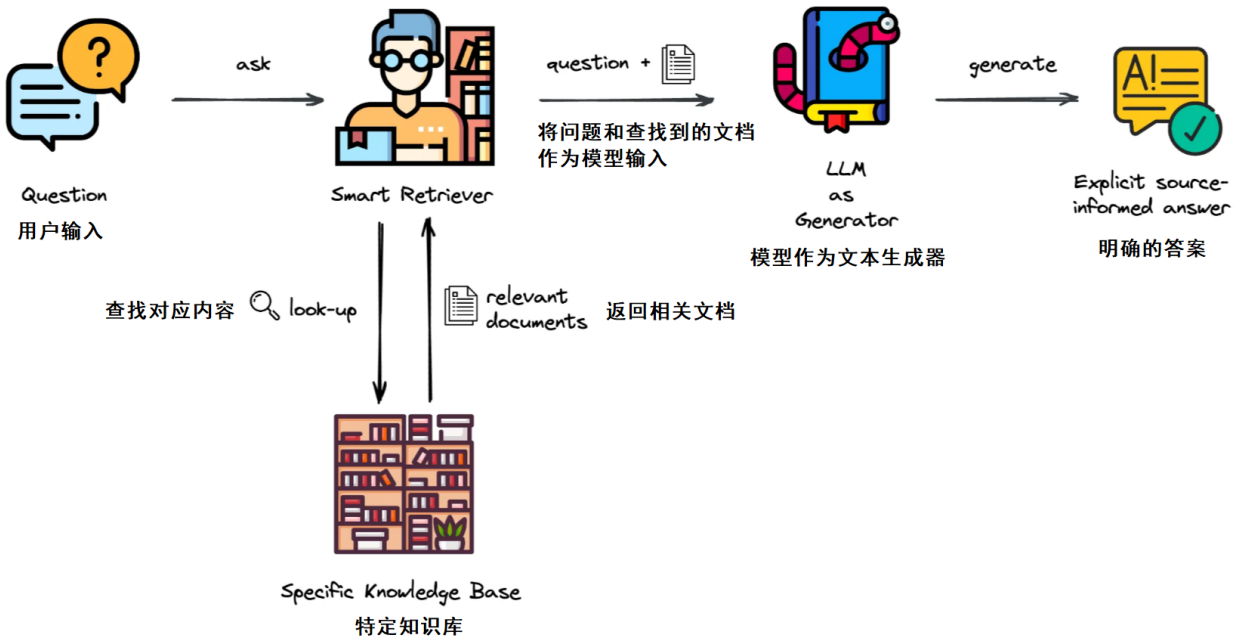
6. 对检索后的结果使用重排 (reranking) 技术:

- 重新对检索结果进行排序，对于检索式生成的方法有一定作用。重新排序可以使用一个微调过的BERT模型，也可以是使用大语言模型对检索结果进行 Zero-Shot 评估后重新排序，而这类方法还需要进一步的研究。

第五节

总结

总结



- **基于LLM的检索式生成模型的核心组成部分：**

1. 检索器 Retriever
2. 生成式语言模型 LLM
3. 知识库（数据库） knowledge Base

- **两种主流的检索器算法：**

1. BM25 (稀疏检索)
2. DPR（向量检索）

- **三种检索式生成方法**：

1. 输入增强 （input augmentation）
2. 中间融合 （Intermediate fusion）
2. 插值输出 （output interpolation ）

- **三种不同的训练方式：**

1. 独立训练 (Independent Training)
2. 序列训练（Sequential Training）
3. 联合训练 （Joint Training）

参考资料

参考博文：
https://blog.ml6.eu/leveraging-llms-on-your-domain-specific-knowledge-base-4441c8837b47
https://docs.llamaindex.ai/en/stable/examples/retrievers/bm25_retriever/
https://blog.csdn.net/huanxingchen1/article/details/130241071
https://sh-tsang.medium.com/brief-review-dpr-dense-passage-retrieval-for-open-domain-question-answering-9323cd8f85c4
https://blog.51cto.com/u_16213301/11832863
https://zhuanlan.zhihu.com/p/360575335

参考论文：
A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models (Wenqi Fan etal,KDD 2024)
Reliable, Adaptable, and Attributable Language Models with Retrieval (Akari asai, etal, Arxiv)
In-Context Retrieval-Augmented Language Models (Ori Ram, ACL 2023 Findings)
Generalization through Memorization: Nearest Neighbor Language Models (Urvashi Khandelwal, ICLR 2020)
Training Language Models with Memory Augmentation (Zhong et al., EMNLP 2022)
Nonparametric masked language modeling. （Min S et al., ACL 2023 Findings）
Copy is all you need (T Lan et al., ICLR 2023)
RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation (Xu et al, ICLR 2024)
Improving language models by retrieving from trillions of tokens (Sebastian et al, Deepmind)
Dense Passage Retrieval for Open-Domain Question Answering (Vladimir et al, EMNLP 2020)

谢谢聆听！

Q&A