

RAG 技术详解与实战应用

第6讲：检索更准：

RAG召回效果优化的底层逻辑与技巧



目录



1. 上节回顾



2. RAG效果评测



3. 提升RAG系统的召回率



4. 多路召回的RAG



自定义 Reader 概述

- **Reader 功能：**Reader模块用于从各种格式的知识库中读取内容，并将其转换为统一的格式供检索模块使用。
- **支持情况：**默认支持多种文档格式，如pdf、doc、hwp、ppt、ipynb、epub、markdown、mbox、csv、excel、image、MP3、MP4。
- **自定义原因：**当知识库中出现LazyLLM不支持的文档格式或现有Reader的输出格式不符合需求时，可以自定义Reader满足特定需求。

自定义 Reader 方法

- **HTML Reader 示例：**用 BeautifulSoup 等解析 HTML 提取文本并实例化。
- **注册方式：**通过Document类的add_reader方法将自定义的Reader注册到LazyLLM中，使其能够被系统识别和使用。
- **构建基于HTML文档的RAG应用：**使用自定义的Reader实现一个简单的RAG应用，展示其在问答任务中的效果。

复杂场景实现

- **多级注册：**key-value 方式注册，支持多种匹配规则，优先级从高到低为“通过对象注册的Reader” > “通过类注册的Reader” > “LazyLLM默认提供的Reader”。
- **基于类 Reader：**继承 ReaderBase 并实现_load_data接口来定义自定义Reader。
- **复杂 PDF 解析：**使用magic-pdf工具包进行PDF文档解析，支持提取文本、图像和表格等多种内容，并通过自定义的UnionPdfReader实现高效解析。



目录



1. 上节回顾



2. RAG效果评测



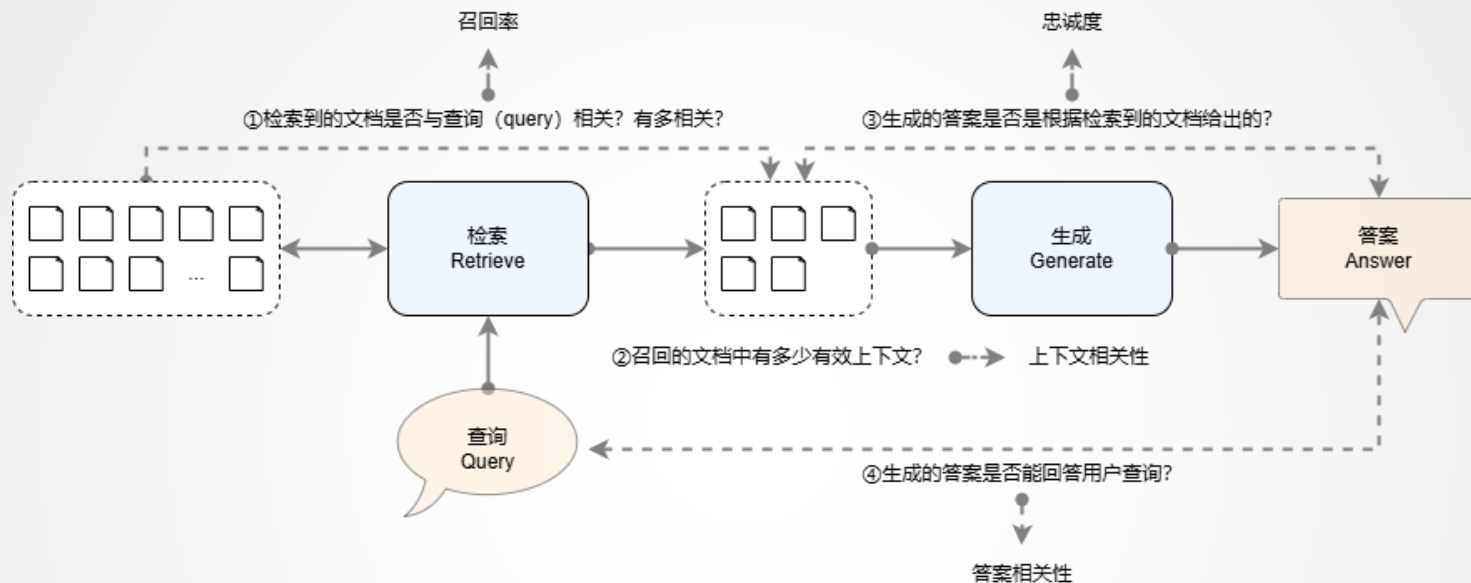
3. 提升RAG系统的召回率



4. 多路召回的RAG



RAG 系统通过检索与生成协同工作，引入外部知识以减少幻觉，提升回答质量。其效果主要取决于四个关键环节：



1. **召回率**：检索器返回的文档中，被检索出的相关文档占所有相关文档的比例。
2. **上下文有效性 (准确率)**：召回文档中有用信息的占比，噪声越少，生成越准确。
3. **生成忠诚度**：生成内容是否忠于上下文，避免脱离原文编造答案。
4. **答案相关性**：最终输出是否真正回应用户提问。

优化以上四点，可系统性提升 RAG 的整体表现。



在 RAG 实践中，评估检索质量常用两个指标：召回率和上下文相关性，两者数值越高越好。高召回率与高相关性有助于生成更贴近用户问题的答案。

1. 召回率

召回率表示被检索出的相关文档占所有相关文档的比例，用于衡量检索器的查全能力。其公式为：

$$Recall = \frac{TP}{TP + FN}$$

- 其中，TP 表示成功召回的相关文档数，FN 表示未被召回的相关文档数。

例如文档库中有5篇文档是与当前查询相关的，而检索组件检索只检索到了3篇，则召回率为 $3/5=0.6$ ，实际应用中，召回率越高，对生成组件提供的相关证据就越多，有助于提升RAG的总体效果。

2. 上下文相关性（精确率）

上述召回率是文档块级别的评估方法，此外还可更细粒度的方法评估检索器，如 RAGAS 提出的上下文相关性（context relevance）。它在检索文档中提取与查询相关的句子，并在句子级别计算精度：

$$CR = \frac{\text{与查询相关的句子总数}}{\text{召回文档中的句子总数}}$$

上下文相关性衡量检索结果中与查询相关句子的占比，值越高说明提供的参考信息越有效，便于生成组件提供关键内容。例如检索到的文档中共10句，仅4句与查询相关，则上下文相关性为 $4/10 = 0.4$ 。

RAG的生成组件主要是通过检索组件给出的文档进行答案生成，主要从**忠诚度 (faithfulness)** 和**答案相关性 (Answer Relevance)** 两个维度进行评估。其中忠诚度评估的是生成组件的答案与检索组件给出的上下文相关，对对抗大模型幻觉至关重要，而答案相关性则需要评估生成组件给出的答案是否能够准确响应问题。

1. 忠诚度

当生成组件给出的答案是可以在上下文中找到出处的，则称答案 a 是忠诚于上下文 c 的。为了构造客观计算指标，通常会按照如下方式流程进行：

- (1) 利用LLM从答案中抽取一系列语句，这一步的目的是将长难句精简为多个精简准确的断言 S（或命题）；
- (2) 将上一步提取的断言 S 和检索器给出的上下文一同输入给LLM，让LLM推断这些断言是否来自检索上下文；
- (3) 计算最终得分

$$F = \frac{|V|}{|S|}$$

其中|V|是来源于检索上下文的断言个数，|S|是所有断言个数。该值越大，说明该系统的忠诚度越高。



生成组件评估



下面一起看一个例子：

Query：地球上最高的山是哪座？

Context（上下文）：珠穆朗玛峰（Mount Everest）是地球上最高的山峰，海拔约 8848.86 米，位于尼泊尔和中国西藏的边界。

高度忠诚的答案：地球上最高的山是珠穆朗玛峰，海拔 8848.86 米。

低忠诚度答案：地球上最高的山是珠穆朗玛峰，海拔 5895 米。

我们以其中的低忠诚度答案为例，构造断言

断言1：地球上最高的山是珠穆朗玛峰。

断言2：地球上最高的山海拔是5895米。

此时大模型应输出的答案为：

断言1：是。

断言2：不是。

因此针对这个问题的答案，该系统的忠诚度得分为**0.5**。



生成组件评估

2. 答案相关性



答案相关性（Answer Relevance, AR）衡量的是对用户查询生成的答案的质量，即答案与用户问题的相关性。检查答案是否完整，是否包含冗余信息。为了评估这一点，通过大模型生成N个问题，如果提供的答案与原始问题相关，则问题应该与原始问题高度相似。其流程可分解为：

- (1) 给定答案，让LLM生成该答案可能对应的N个问题，N一般在3到4之间；
- (2) 对所有可能的问题进行向量嵌入；
- (3) 计算潜在问题向量 q_i 和原始查询向量 q 的相似度：

$$AR = \frac{1}{N} \sum_{i=1}^N sim(q, q_i)$$

其中sim可以是余弦相似度等向量相似度计算方法。同样针对“地球上最高的山是珠穆朗玛峰，海拔 8848.86 米”的答案，利用大模型生成答案，得到如下三个可能的问题

问题1：地球上最高的山是哪座？

问题2：珠穆朗玛峰的海拔是多少？

问题3：世界上哪座山的海拔最高？

然后将上述三个问题的向量与查询向量 q ，即“地球上最高的山是哪座？”，计算平均相似度。



评测示例



对于查询：“深度学习的应用有哪些？”，文档库内容（其中✅是与查询相关的文档，❌为不相关）：

✅ 深度学习在计算机视觉中广泛应用，如图像分类、目标检测和自动驾驶。它可以提高图像识别的准确率，使得自动驾驶车辆能够实时感知周围环境。

❌ 传统机器学习依赖手工特征工程，而深度学习自动学习特征。相比之下，深度神经网络能够在海量数据中自动提取有效特征，减少人工干预。

✅ 深度学习可以用于医学影像分析，辅助医生诊断疾病。比如，AI 可以自动识别 X 光片中的异常，提高疾病早期发现的准确率。

✅ 深度学习在推荐系统中，如 Netflix 和淘宝推荐算法，起到了重要作用。它能分析用户的浏览和购买记录，提供个性化的内容推荐。

❌ 深度学习的计算复杂度较高，需要 GPU 进行加速。高性能计算设备的支持使得深度学习能够处理大规模数据，提高训练速度。

✅ 自然语言处理是深度学习的重要应用，包括机器翻译和对话系统。基于深度学习的 NLP 技术让翻译系统更加自然流畅，并提升智能助手的理解能力。

✅ 金融行业也受益于深度学习，特别是在欺诈检测和股票市场预测方面。例如，AI 可分析交易模式，识别异常行为以防止欺诈。

❌ 循环神经网络（RNN）适用于时间序列预测。它可以用于金融市场分析，预测股票价格走势。

❌ 卷积神经网络（CNN）在图像处理方面效果显著。它能够自动提取图像的边缘、形状等特征，提高计算机视觉任务的表现。

✅ 强化学习是深度学习的一部分，已应用于游戏 AI，如 AlphaGo。通过不断自我对弈，AI 能够学习最佳策略，在围棋等游戏中超越人类高手。

评测示例



对于查询: "深度学习的应用有哪些?", 关键词检索召回的句子有如下 5 句:

- ✅ 深度学习在计算机视觉中广泛应用, 如图像分类、目标检测和自动驾驶。它可以提高图像识别的准确率, 使得自动驾驶车辆能够实时感知周围环境。
- ❌ 传统机器学习依赖手工特征工程, 而深度学习自动学习特征。相比之下, 深度神经网络能够在海量数据中自动提取有效特征, 减少人工干预。
- ✅ 深度学习可以用于医学影像分析, 辅助医生诊断疾病。比如, AI 可以自动识别 X 光片中的异常, 提高疾病早期发现的准确率。
- ✅ 深度学习在推荐系统中, 如 Netflix 和淘宝推荐算法, 起到了重要作用。它能分析用户的浏览和购买记录, 提供个性化的内容推荐。
- ❌ 深度学习的计算复杂度较高, 需要 GPU 进行加速。高性能计算设备的支持使得深度学习能够处理大规模数据, 提高训练速度。

检索召回率: 有 3 句是正确的 (真正相关), 2 句是错误的 (不属于应用), 文档库中与深度学习应用相关的句子一共有 6 句, 因此召回率是 $3/6=0.5$ 。

检索上下文相关性: 在召回的 5 句话中, 其中 3 个句子是与查询相关的, 因此其上下文相关性是 $3/5=0.6$ 。



对于“深度学习的应用有哪些？”和检索到的上下文，模型 (internlm2-7b) 给出答案：

深度学习在多个领域有广泛的应用，以下是一些关键的应用领域：

1. 计算机视觉：深度学习在图像分类、目标检测和自动驾驶等计算机视觉任务中发挥重要作用。通过深度神经网络，深度学习能够从大量数据中自动提取和识别图像中的特征，提高自动驾驶车辆对周围环境的感知能力，从而提升安全性。
2. 医疗健康：在医学影像分析领域，深度学习可以帮助医生快速准确地诊断疾病。例如，AI能够自动识别X光片中的异常，从而提高疾病早期发现的准确率，有助于更早地进行干预和治疗。
3. 推荐系统：深度学习在推荐系统中被广泛应用，如Netflix和淘宝的推荐算法。通过分析用户的浏览和购买记录，深度学习能够个性化地推荐内容，提高用户满意度和购买转化率。
4. 自然语言处理：在文本分析、语音识别、机器翻译等方面，深度学习也有显著的应用。通过学习大量语言数据，深度学习模型能够理解和生成自然语言，为智能客服、智能助理等应用提供支持。
5. 金融服务：深度学习在欺诈检测、风险评估和投资策略等方面有所应用。通过分析大量的金融数据，深度学习可以帮助金融机构更好地识别风险，优化投资组合，提升业务效率。
6. 科学研究：深度学习在科学研究中也有广泛的应用，如在天文学、生物学、化学等领域中用于数据分析和模式识别。通过深度学习，科学家们可以更快速地处理和分析复杂的数据，从而推动科学研究的进展。
7. 游戏开发：深度学习在游戏AI中扮演重要角色，能够帮助游戏角色更好地理解游戏规则和玩家行为，提供更智能的游戏体验。
8. 工业自动化：在制造业和工业自动化领域，深度学习可用于预测设备故障、优化生产流程和提高产品质量。通过分析传感器数据，深度学习模型能够提前预警故障，减少停机时间和生产成本。

总之，深度学习的应用范围非常广泛，它正在改变各个行业的运作方式，并提供更高效、更智能的解决方案。随着技术的不断进步和算法的优化，深度学习的应用前景仍然广阔。



示例

利用大模型抽取得到如下10个断言（由ChatGPT抽取），并推断这些断言是否来自检索上下文：

- ✓ "深度学习在计算机视觉任务中发挥重要作用。"
- ✓ "深度学习能够提高自动驾驶车辆对周围环境的感知能力。"
- ✓ "深度学习可以帮助医生快速准确地诊断疾病。"
- ✓ "深度学习能够自动识别 X 光片中的异常，提高疾病早期发现的准确率。"
- ✓ "深度学习在推荐系统中被广泛应用，如 Netflix 和淘宝的推荐算法。"
- ✓ "深度学习用于自然语言处理任务，如文本分析、语音识别和机器翻译。"
- ✗ "深度学习可以帮助金融机构识别风险并优化投资策略。"
- ✗ "深度学习在科学研究中用于数据处理和模式识别。"
- ✗ "深度学习可以优化工业自动化中的生产流程和产品质量。"
- ✗ "深度学习正在改变各个行业的运作方式，并提供更高效、更方案。"

计算忠诚度得到：

$$\bullet \quad 6 / 10 = 0.6$$

LLM 根据大模型答案生成可能的问题：

- 深度学习在哪些领域有重要应用？

计算原问题"深度学习的应用有哪些？"和LLM生成的最可能的问题"深度学习在哪些领域有重要应用？"之间的相似度（此处以余弦相似度为例）：

- 计算结果为 0.8 。

| 召回率 | 上下文相关性 | 忠诚度 | 答案相关性 |
|-----|--------|-----|-------|
| 0.5 | 0.6 | 0.6 | 0.8 |



目录



1. 上节回顾



2. RAG效果评测



3. 提升RAG系统的召回率



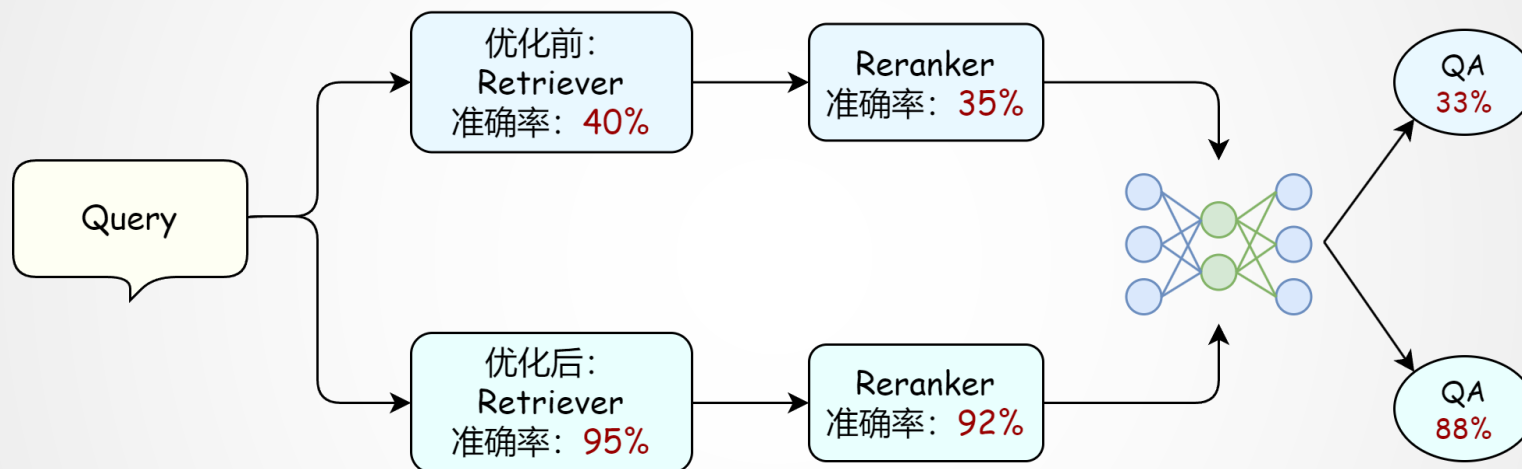
4. 多路召回的RAG



RAG的效果，重点在R

示例案例：某企业内部知识问答系统优化效果

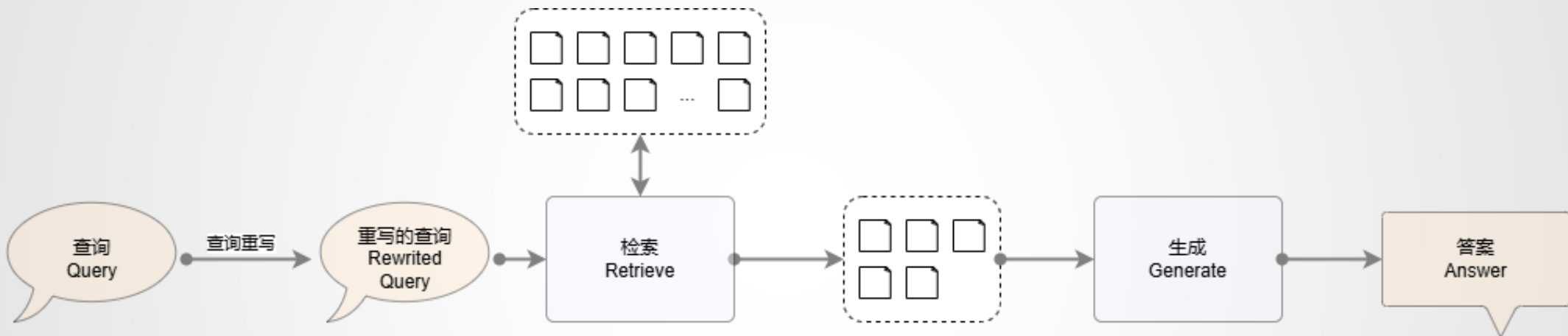
背景：某企业部署了基于RAG的内部知识问答系统，用于员工快速查询政策与流程。但初期效果不佳，用户反馈准确率低。



结论：通过引入多路召回、优化重排序策略与微调生成模型，RAG 系统各阶段准确率显著提升，特别是召回准确率从 **40%** 提升至 **95%**，带动后续重排准确率提升至 92%，问答准确率提升至 88%。这表明：**召回阶段是提升整个 RAG 系统表现的关键环节**，高质量的召回不仅提高了文档相关性，也为生成组件提供了更准确、丰富的上下文，从而显著提升最终答案的准确率与可靠性。

下面介绍几种简单的查询重写策略。

1. 扩写查询



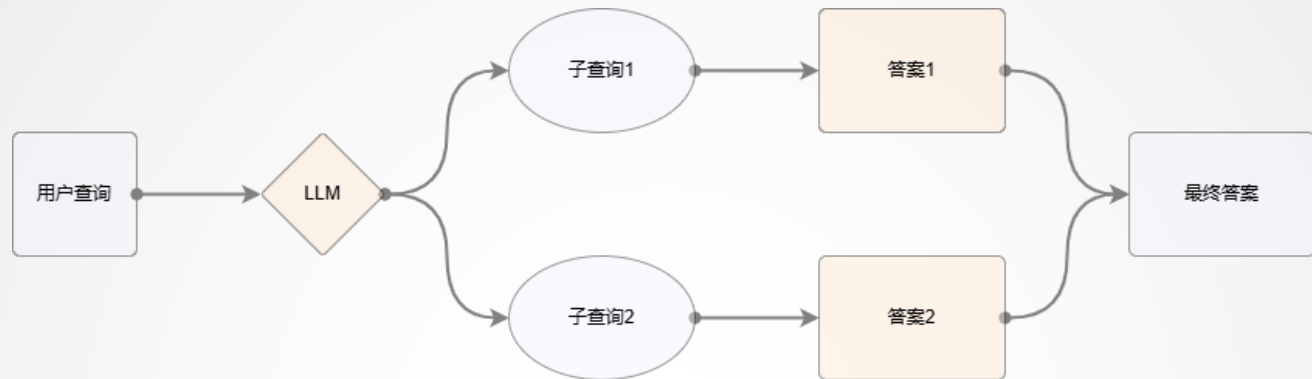
用户查询常常存在歧义或信息不足，影响系统理解和检索效果。查询扩写（Query Expansion）通过补充或优化原始查询，使其更清晰具体，从而提升检索准确率和召回率。

- 常见策略包括：**同义词扩展**（丰富表达）、**上下文补充**（利用历史对话消除歧义）以及**问题模板转换**（结构化查询以匹配检索系统偏好）。这些方法有助于系统更好理解用户意图，提升检索效果。



查询重写策略

2. 子问题查询



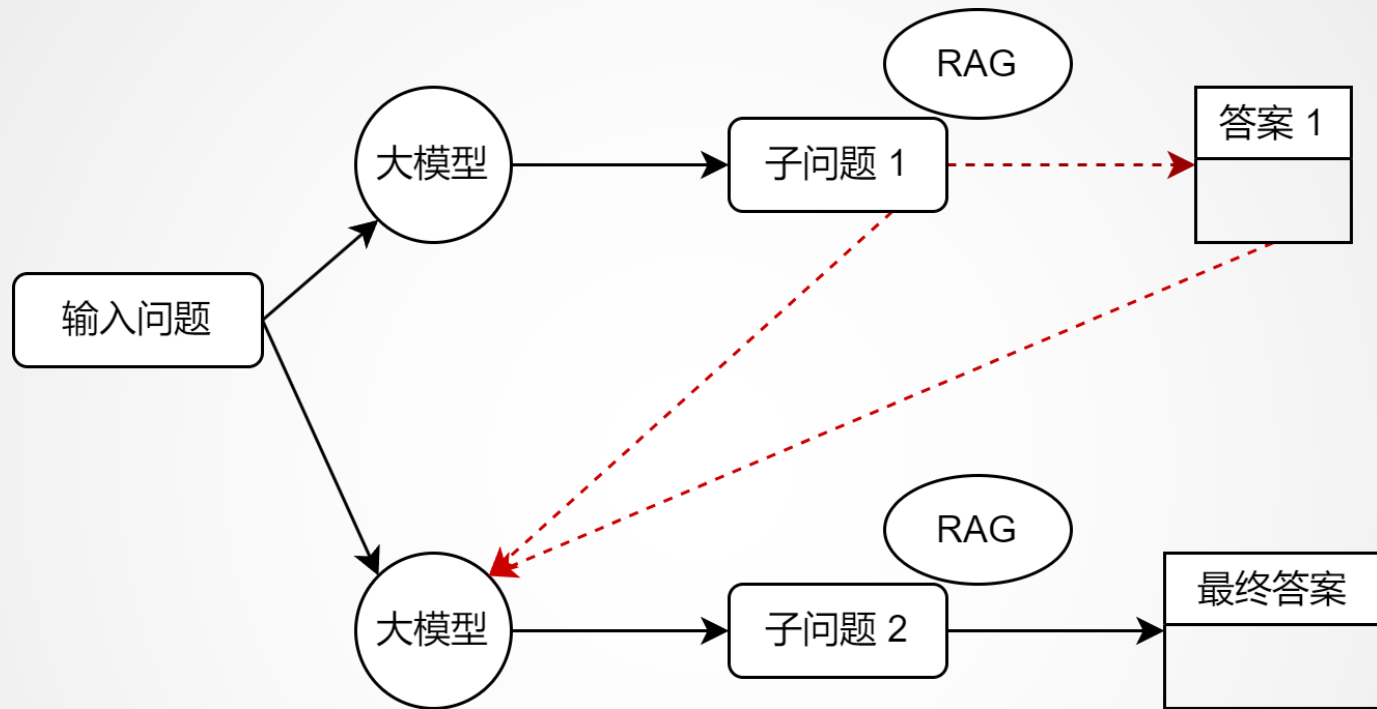
子问题查询策略的核心思想是生成并提出与主问题相关的子问题，从而更好地理解 and 回答主问题。子问题通常更具体，能够帮助系统更深入地理解主问题，通常用于比较抽象的问题。简单的子查询构造流程如上流程图：

- (1) 通过 LLM 从用户查询中生成多个子问题，例如对原始问题“RAG是什么？”，可以进行如下子查询分解：“RAG的定义是什么？”，“RAG的特点是什么？”，“RAG的原理是什么？”等，子查询构造可以为问题提供多种角度，从而使答案更加具体更加全面；
- (2) 每个子问题都经过RAG获取相关答案；
- (3) 将所有问题合并以获得最终答案。

！注意：子问题策略虽可提升召回多样性，但也增加系统开销，需结合工程优化确保响应速度。



3. 多步骤查询

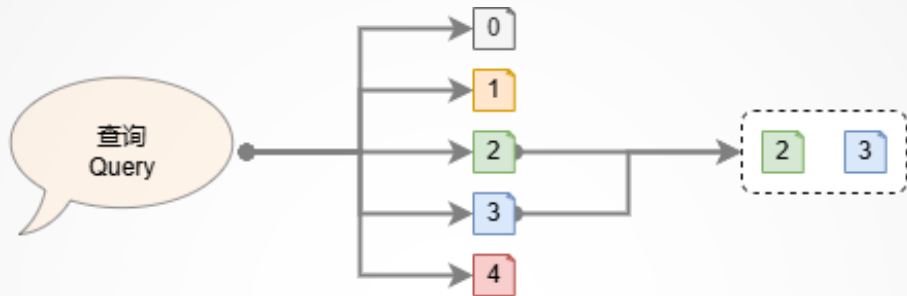


RAG 中多步骤查询的核心思想是利用大模型将复杂查询拆解为多个子问题，首先生成子问题1并获取答案，再将答案与上下文一同输入大模型生成子问题2等等，层层推进，直至得到最终答案，适用于需要多轮推理或信息串联的场景。简单的多步骤查询构造流程如上流程图。



优化检索策略

检索是指根据用户需求，在文档库中寻找相关信息的技术。核心在于衡量查询与文档的相关度，从而选出最匹配的内容。下图展示了一个简单流程：查询与五个文档比对后，返回与其相关性最高的id 为 2 和 3的两个文档。



要提升召回质量，可从以下三方面优化：

1. **被检索对象（节点组）**：将长文档拆分为更具针对性的子片段，避免主题分散影响召回效果。这种策略在 RAG 中被称为“文档解析”，每类解析规则形成一个“节点组”，适配不同任务需求。
2. **被检索对象的表征方式（原文 vs 向量化）**：选择使用原文或向量表示。原文适合快速匹配关键词，向量化则保留语义信息，适合语义检索。
3. **检索对象的具体检索方式（相似度）**：根据表征方式选择合适的相似度算法，如原文匹配适合关键词重合度，向量化则配合余弦相似度等方法。

合理组合以上策略，有助于显著提升召回质量，为生成环节提供更相关的上下文。



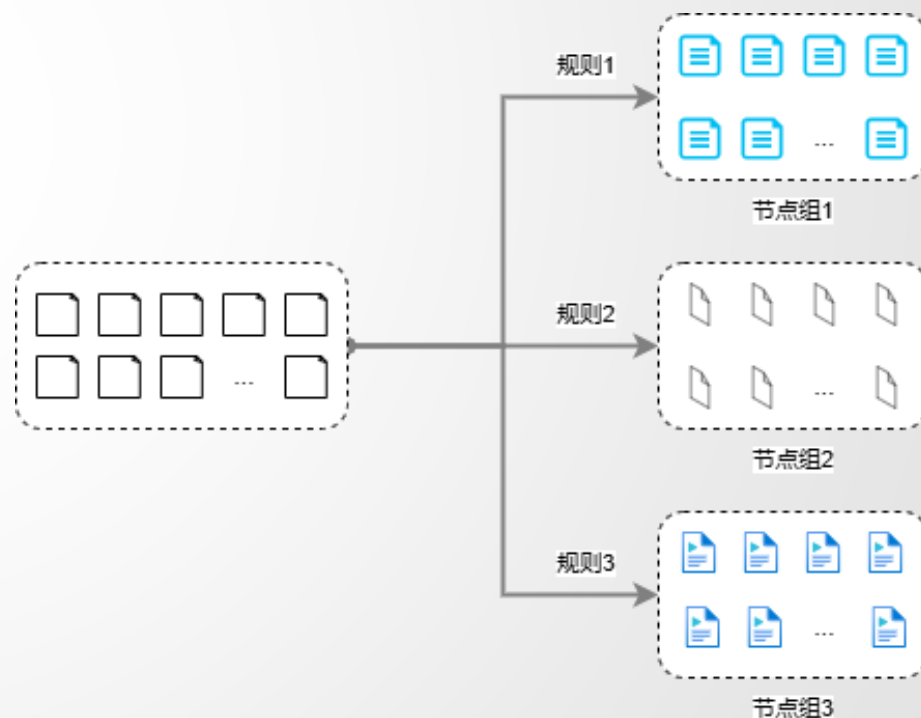
优化检索策略 – 节点组

本节介绍 RAG 系统中两种常见的节点构造方法：**基于分块的策略**和**基于语义提取的策略**。

- 前者包括固定**大小分块**和**递归分块**，前者简单但可能破坏语义，后者利用分隔符更好保留语义完整性。
- 后者则通过摘要、关键词、问答对等方式提取语义更突出的节点，提升检索效率和效果。

通过本节内容，您将了解不同节点策略对召回的影响，并掌握如何根据任务选择合适方式，提高检索准确性与效率。

节点组是检索组件执行检索操作的对象，一次检索只针对一个节点组进行。右图展示了如何从原文构造节点组：针对文档集中的所有文档，通过应用不同的解析规则得到几个不同的节点组。不同节点组中的内容互不相同，例如通过不同的长度约束对文档分块得到不同粒度的文档块节点组或通过摘要提取等语义处理手段得到主题相同但文本有所差异的节点组。



优化检索策略 – 节点组切分方式

(1) 固定大小分块 (Fixed Size Chunking)

固定大小分块是最常见的分块方式，将文本按**固定长度**切块，通常以 **token** 为单位。虽然这种方法简单高效，但缺乏对上下文的理解，容易打断句意。为改善这一问题，可采用重叠分块，让相邻块共享部分内容，增强连续性。

在我的后园，可以看见墙外有两株树，一株是枣树，还有一株也是枣树。

这上面的夜的天空，奇怪而高，我生平没有见过这样奇怪而高的天空。他仿佛要离开人间而去，使人们仰面不再看见。然而现在却非常之蓝，闪闪地映着几十个星星的眼，冷眼。他的口角上现出微笑，似乎自以为大有深意，而将繁霜洒在我的园里的野花草上。

我不知道那些花草真叫什么名字，人们叫他们什么名字。我记得有一种开过极细小的粉红花，现在还开着，但是更极细小了，她在冷的夜气中，瑟缩地做梦，梦见春的到来，梦见秋的到来，梦见瘦的诗人将眼泪擦在她最末的花瓣上，告诉她秋虽然来，冬虽然来，而此后接着还是春，蝴蝶乱飞，蜜蜂都唱起春词来了。

- 左图展示了固定大小分块的示例，每块长度为25，重叠5个 token。可以看到，“一株是枣树”这句话被拆开，前一块没提到枣树位置，后一块也缺少完整信息，导致上下文断裂，影响理解和检索效果。

其中每种颜色代表一个块，绿色部分为相邻块间的重叠部分，这个例子选择的固定大小是25，重叠数是5。



优化检索策略 – 节点组切分方式



(2) 递归分块 (Recursive Chunking)

递归分块不同于固定大小分块，它优先按**段落等**主分隔符切分，若仍过长，再按句子等次级分隔符继续细分。这样能在保持语义完整的前提下处理长文本，提升上下文理解与生成质量，避免信息被割裂。

在我的后园，可以看见墙外有两株树，一株是枣树，还有一株也是枣树。

这上面的夜的天空，奇怪而高，我生平没有见过这样奇怪而高的天空。他仿佛要离开人间而去，使人们仰面不再看见。然而现在却非常之蓝，闪闪地映着几十个星星的眼，冷眼。他的口角上现出微笑，似乎自以为大有深意，而将繁霜洒在我的园里的野花草上。

我不知道那些花草真叫什么名字，人们叫他们什么名字。我记得有一种开过极细小的粉红花，现在还开着，但是更极细小了，她在冷的夜气中，瑟缩地做梦，梦见春的到来，梦见秋的到来，梦见瘦的诗人将眼泪擦在她最末的花瓣上，告诉她秋虽然来，冬虽然来，而此后接着还是春，蝴蝶乱飞，蜜蜂都唱起春词来了。

- 例如上图例子展示了以换行符为分隔符的例子，与固定大小分块相比，该方法可以保留一定程度上的语义连贯性，保证每个块的语义内容是完整的。



优化检索策略 – 节点组切分方式

(3) 语义分块 (Semantic chunking)

语义分块是一种基于文本语义信息进行的分块方法，它不依赖于简单的字符或词的长度，而是**根据文本中的语义单元进行分割**。具体来说，它首先将文本通过向量化，然后计算嵌入之间的余弦距离，将特征距离近的嵌入组合在一起构成一个块。通过这种方式，文本被拆分成具有语义完整性和连贯性的块，每个块内的内容通常围绕一个特定的主题或意思展开。

- 例如右侧例子对文档进行了语义分块，可见将标题和对应的内容分到了一个块中，相比于直接基于换行符（“\n”）进行分块，保证了一定的语义连续性。



LazyLLM: 低代码构建多Agent大模型应用的开发工具 ## 一、简介
LazyLLM是一款低代码构建**多Agent**大模型应用的开发工具，协助开发者用极低的成本构建复杂的AI应用，并可以持续的迭代优化效果。

二、特性 **便捷的AI应用组装流程**：即使您不了解大模型，您仍然可以像搭积木一样，借助我们内置的数据流和功能模块，轻松组建包含多个Agent的AI应用。

****跨平台兼容****：无需修改代码，即可一键切换IaaS平台，目前兼容裸金属服务器、开发机、Slurm集群、公有云等。

这使得开发中的应用可以无缝迁移到其他IaaS平台，大大减少了代码修改的工作量。

****支持网格搜索参数优化****：根据用户配置，自动尝试不同的基模型、召回策略和微调参数，对应用进行评测和优化。

这使得超参数调优过程无需对应用代码进行大量侵入式修改，提高了调优效率，帮助用户快速找到最佳配置。

****高效的模型微调****：支持对应用中的模型进行微调，持续提升应用效果。

根据微调场景，自动选择最佳的微调框架和模型切分策略。

这不仅简化了模型迭代的维护工作，还让算法研究员能够将更多精力集中在算法和数据迭代上，而无需处理繁琐的工程化任务。

优化检索策略 – 节点组切分方式



(4) 基于文档的分块 (Document-based chunking)

基于文档的分块在大规模文本处理和信息检索系统中较为常见。与传统的基于字符或词的分块方法不同，该方法根据**文档中的自然划分（例如标题或章节）**创建分块，每个文档作为一个独立的处理单元，这样的分块方式有助于保持文本的整体语境和信息结构。这种方法对于 HTML、Markdown 或代码文件等结构化数据特别有效，但当数据缺乏明确的结构元素时用处不大。

- 例如右侧示例，相比于前面示例，对标题和内容进行了较好的区分，保持了原文本的结构性没有被破坏。

LazyLLM: 低代码构建多Agent大模型应用的开发工具

一、简介

LazyLLM是一款低代码构建****多Agent****大模型应用的开发工具，协助开发者用极低的成本构建复杂的AI应用，并可以持续的迭代优化效果。

二、特性

****便捷的AI应用组装流程****：即使您不了解大模型，您仍然可以像搭积木一样，借助我们内置的数据流和功能模块，轻松组建包含多个Agent的AI应用。

****跨平台兼容****：无需修改代码，即可一键切换IaaS平台，目前兼容裸金属服务器、开发机、Slurm集群、公有云等。这使得开发中的应用可以无缝迁移到其他IaaS平台，大大减少了代码修改的工作量。

****支持网格搜索参数优化****：根据用户配置，自动尝试不同的基模型、召回策略和微调参数，对应用进行评测和优化。这使得超参数调优过程无需对应用代码进行大量侵入式修改，提高了调优效率，帮助用户快速找到最佳配置。

****高效的模型微调****：支持对应用中的模型进行微调，持续提升应用效果。根据微调场景，自动选择最佳的微调框架和模型切分策略。这不仅简化了模型迭代的维护工作，还让算法研究员能够将更多精力集中在算法和数据迭代上，而无需处理繁琐的工程化任务。



优化检索策略 – 特殊节点组



下面通过一个打印机售后服务的场景，简单解释摘要、关键词、问答对节点组的优势：


(1) 关键词节点组：

如用户问“**如何清理打印机墨盒？**”，系统通过提取关键词“**墨盒**”、“**清洁**”等，根据这些关键词快速定位相关段落，从而将这些相关段落和用户提问送给大模型进行回答，有效提升检索效率。

(2) 摘要节点组：

面对如“**如何连接打印机到Wi-Fi？**”的问题，仅靠关键词可能引入干扰，而通过生成文档摘要，可直接提取核心信息，准确召回相关内容。例如，**打印机连接设置相关内容部分生成摘要后对应文档部分的摘要信息中必然包含打印机的Wi-Fi连接信息**，在获取摘要对应的原文后，将其与用户问题一并传入大模型，实现基于上下文的回答。

(3) 预置QA对节点组：

如用户问“**我的打印机提示纸张卡住，我该怎么办？**”，相关处理步骤可能分散在多个段落，通过构建标准化的问答对，可直接提供完整答案，提升响应效率和体验。例如：我们召回了与用户问题相似的问题“**打印机显示纸张卡住，该如何处理？**”、“**遇到打印机卡纸问题，我该如何操作？**”、“**如何处理打印机出现的卡纸提示？**”，接着将召回的问题和答案以及用户提问一起传给大模型，完成用户问答。

向量化（也称为嵌入，embedding）是一种常见的文本表示方式。向量化通过模型将文本转为高维向量，用于语义相似度检索。

| 比较维度 | 基于原文的检索 | 语义检索 |
|------|--|--|
| 优势 | <ul style="list-style-type: none">- 精准匹配关键词- 可解释性强，结果直观易理解- 计算成本低，查询快、存储省 | <ul style="list-style-type: none">- 具备语义理解能力，能识别同义词与相关概念- 适用于复杂查询，如长句或问题式提问- 提升召回率，支持措辞多样的表达方式 |
| 适用场景 | <ul style="list-style-type: none">- 结构化文档- 法律法规- 技术文档等关键词固定场景 | <ul style="list-style-type: none">- 对话系统- 智能问答- 查询表达形式多样的应用 |
| 劣势 | <ul style="list-style-type: none">- 缺乏语义理解能力- 同义词变体识别困难- 查询词不在文档中则无法召回相关内容 | <ul style="list-style-type: none">- 计算成本高，需要嵌入模型和向量计算- 精度依赖训练数据，专业领域需额外微调 |

在实际应用中，可结合原文检索与语义检索以优化效果。原文检索实现简单，适合直接匹配关键词；语义检索需预先将文本向量化，支持更灵活的查询。下面将简要介绍文本向量化技术。

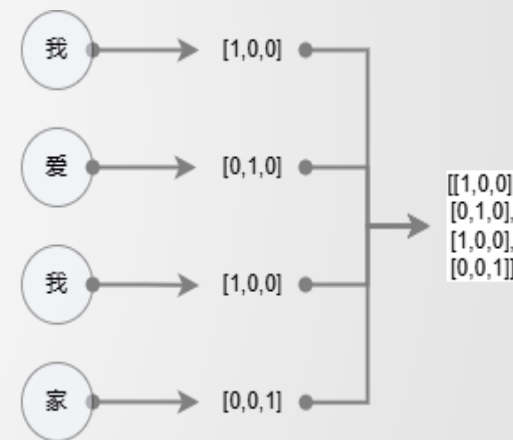
优化检索策略 – 向量化

文本嵌入是一种什么样的技术？

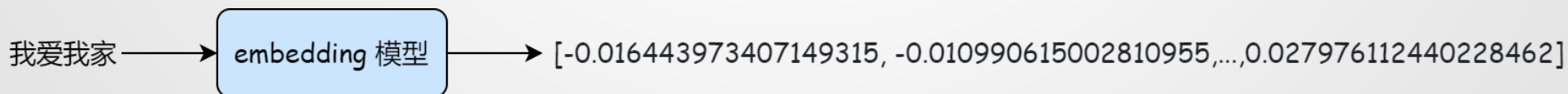
—— 文本嵌入技术最早是通过将单词映射为向量，以便计算机进行相关计算。以“我爱我家”为例，首先建立一个词表（如“我”，“爱”，“家”），然后根据每个字的位置将其标记为1，其他位置为0。这里，“我”对应[1,0,0]，“爱”对应[0,1,0]，“家”对应[0,0,1]。这种方法称为独热编码（one-hot embedding），是一种经典的单词向量化算法，用于将词表中的每个单词表示为一个独特的向量。

- 基于Transformer的文本Embedding模型能有效保留上下文信息，将文本映射为固定维度向量，向量间的相似度反映语义相似。
- 常见的模型有bge和jina系列，用户可以通过hugging face等平台获取或使用在线模型。

| 我 | 爱 | 家 |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |



下图展示了通过神经网络对“我爱我家”四个字进行嵌入的例子，与前面的独热编码相比，其结果维度更高，数值更复杂，但在面向大规模数据的实际应用中的效果往往更好。

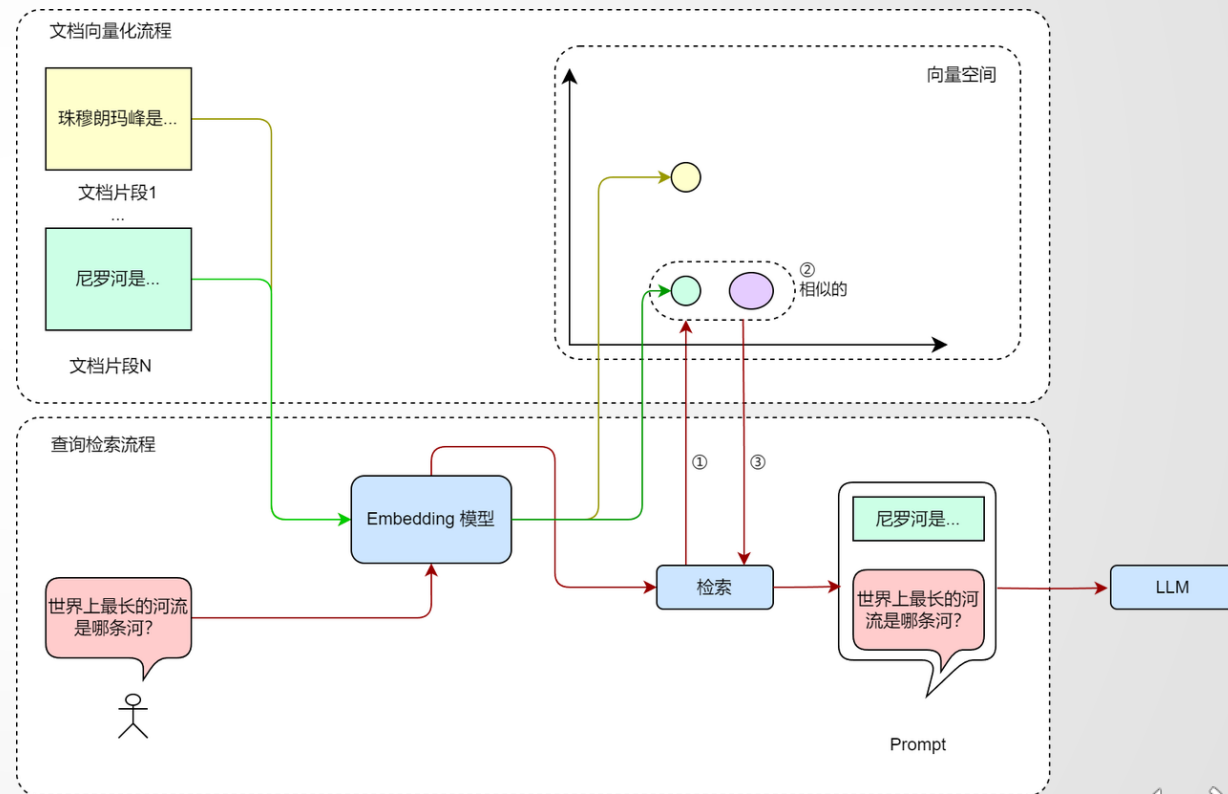


优化检索策略 – 向量检索

如何通过Embedding进行检索？

—— 不难想象利用原文进行检索的算法就是根据用户查询和原文的文档片段进行关键字匹配等，用 Embedding 该如何进行向量检索呢，我们先看下面的图：

- 左图展示了基于语义检索的RAG流程，分为**离线文本向量化**和**在线查询检索**两部分。文档向量化将切片后的文档编码为向量并存入文档库，与查询无关；查询检索则是将用户查询向量化后，计算与文档库向量的相似度，返回相关文档片段供生成模型使用。
- 例如用户查询“世界上最长的河流是哪条河？”，系统将该问题向量化后，在嵌入空间中找到与之最接近的向量，如表示尼罗河信息的文档向量，并返回相应片段给大模型进行回答。



这一流程实现了通过语义匹配进行高效问答的能力。

优化检索策略 – 稠密向量



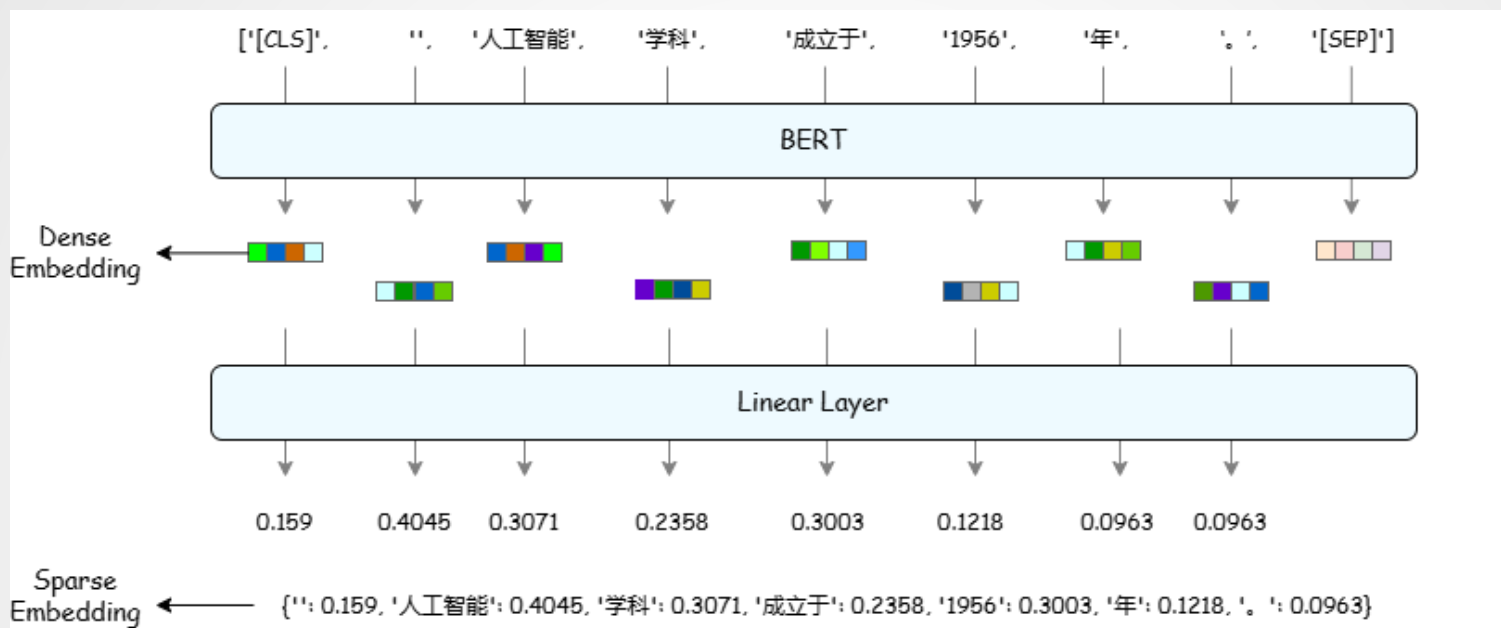
如前所述，向量化能保留文本语义，使相似文本具有更高相似度。常用的向量检索方法有稠密向量检索和稀疏向量检索两类，我们先提前介绍一下稠密向量（默认）。**稠密向量的维度远小于词汇表**，例如 bge-embedding 输入可达 8196 个 token，但输出固定为 1024 维。

- 它的多数元素非零，常由 Word2Vec、BERT、bge-m3、jina 等模型生成。
- 生成方式包括词向量均值池化或提取 BERT 的 [CLS] 向量，能较好表达语义，支持同义词识别和语境理解，即使查询词未出现也可匹配相关内容。
- 但其缺点是计算开销大，且对文本细节把握不如稀疏向量。

```
| [-0.02427494, -0.02393905, 0.02964895, ..., 0.04020474, -0.01293714]
```

稠密向量可读性差、缺乏可解释性，且训练依赖大量数据。向量化过程中，文中的数值、术语、实体等细节可能被忽略，特别是在法律、医学等专业领域，这些关键信息若无法保留，将影响检索的准确性。





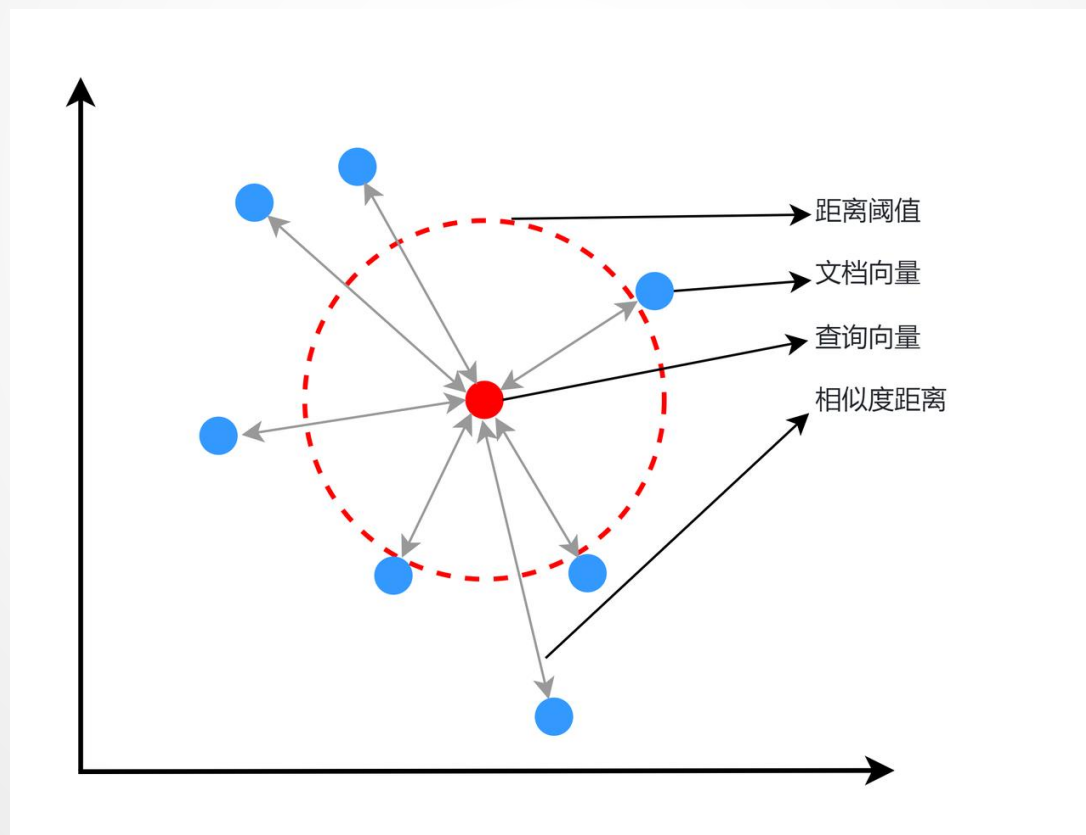
稀疏向量的维度通常等于词表大小（如 bge-m3 为 250002），大多数元素为 0，非零值只出现在实际出现的词上。获取方式包括传统的 TF/TF-IDF 方法和基于深度学习的模型（如 SPLADE、bge-m3），后者结合了语义理解和词级可解释性。相比稠密向量平均化语义，稀疏向量能更好保留词级细节，更适用于需要保留关键信息的场景。由于多数值为 0，稀疏向量通常以 (pos, value) 的形式存储。

```
{'6': 0.159, '81492': 0.4045, '72584': 0.3071, '104220': 0.2358, '189638': 0.3003, '470': 0.1218, '30': 0.0963}
```



优化检索策略 – 在RAG中计算相似度

向量化后的文本嵌入能保留语义信息，因此语义相似的文档向量距离更近。为了找到与用户查询最相关的文档片段，需要选用合适的距离度量函数。常见做法是将查询向量与文档库中所有向量计算相似度，按距离排序选出 top-k 个，或选出在设定阈值范围内的向量作为结果。下图中，**红点为查询向量**，**蓝点为文档向量**，通过比较距离筛选出最相关的点。



优化检索策略 – 相似度计算 内积

- 向量是由n个实数组成的一个n行1列 ($n \times 1$) 或一个1行n列 ($1 \times n$) 的有序数组。向量的内积,也叫向量的点乘、数量积,对两个向量执行点乘运算,就是对这两个向量对应位一一相乘之后求和的操作,点乘的结果是一个标量。设向量 a 和 b 分别
$$\begin{cases} x = [x_1, x_2, \dots, x_n] \\ y = [y_1, y_2, \dots, y_n] \end{cases}$$
- 则 a 和 b 的内积公式为: $x \bullet y = x_1y_1 + x_2y_2 + \dots + x_ny_n$
- 内积表示两个向量在同一方向上的“重合”程度,值越大表示方向越相近,内积为 0 则表示正交。内积常用于稀疏向量的相似度计算。下面以三个 TF-IDF 向量为例进行讲解

A = {1: 0.3, 3: 0.7, 5: 0.5}

B = {1: 0.4, 3: 0.6, 5: 0.2}

C = {2: 0.8, 4: 0.6}

A=(0, 0.3, 0, 0.7, 0, 0.5)

B=(0, 0.4, 0, 0.6, 0, 0.2)

C=(0, 0, 0.8, 0, 0.6, 0)

- 以稀疏表示存储:

- 计算内积:

$$IP(A, B) = (0.3 \times 0.4) + (0.7 \times 0.6) + (0.5 \times 0.2)$$

$$= 0.12 + 0.42 + 0.10$$

$$= 0.64$$

$$IP(A, C) = (0.3 \times 0) + (0 \times 0.8) + (0.7 \times 0) + (0 \times 0.6) + (0.5 \times 0)$$

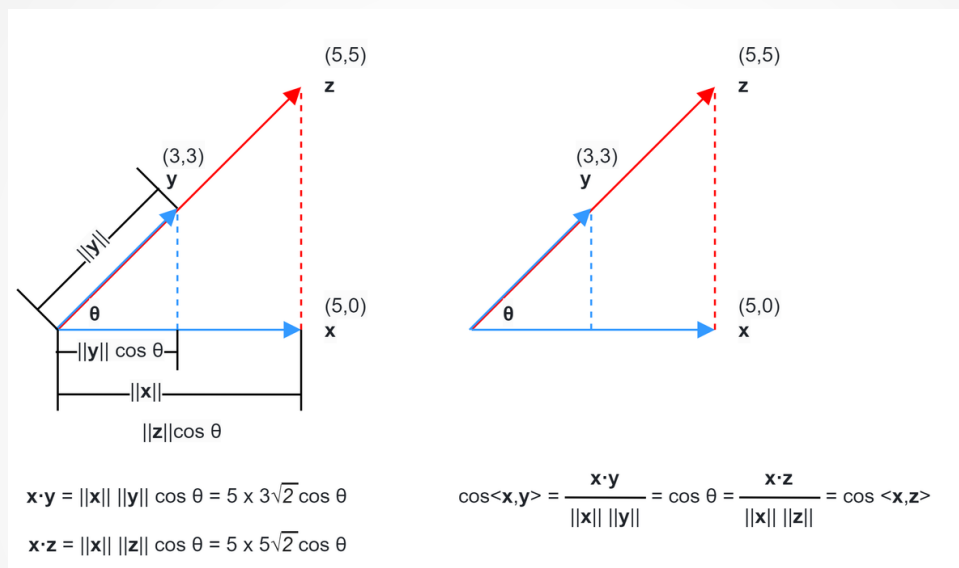
$$= 0$$

可得向量A和B的相似度为 **0.64**, 向量 A 和 C 的相似度为 **0**.



优化检索策略 - 相似度计算 Cosine

余弦相似度是一种标准化的内积度量，它衡量的是两个向量在方向上的相似性，而不考虑它们的长度（即模），如图中的 x , y 与 x , z 虽然长度不相同，但它们的余弦相似度是相等的。



- 余弦相似度的计算方式如下：

$$\text{CosineSimilarity} = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \times \|\vec{y}\|}$$

其中， $\|x\|$ 和 $\|y\|$ 是向量的模。余弦相似度范围为 $[-1, 1]$ ，1 表示方向一致，0 表示正交，-1 表示方向相反。NLP 中余弦相似度是稠密向量常用的相似度计算方式。



优化检索策略 – 相似度计算 BM25

BM25 是传统搜索引擎中常用的基于概率的排序方法，通过词频（TF）、逆文档频率（IDF）和文档长度对文档与查询的相关性进行建模。与余弦或内积不同，BM25 不依赖 embedding，而是直接衡量文本匹配程度：

(1) **词频 (TF)**：词在文档中出现越多，相关性越高。

(2) **逆文档频率 (IDF)**：词越少见，区分能力越强，权重越高。其定义为：

$$IDF(q_i) = \log\left(\frac{N - df(q_i) + 0.5}{df(q_i) + 0.5} + 1.0\right)$$

- 其中，N是文档总数，df(qi) 是包含查询词 qi 的文档数。

(3) **文档长度的归一化**：调整长短文档在词频分布上的影响，避免长文档得分偏高。

已知上述基本思想，记文档为D，查询为Q，f(qi, D)为查询词qi在文档D中的词频，**bm25的计算公式**定义如下：

$$Score(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

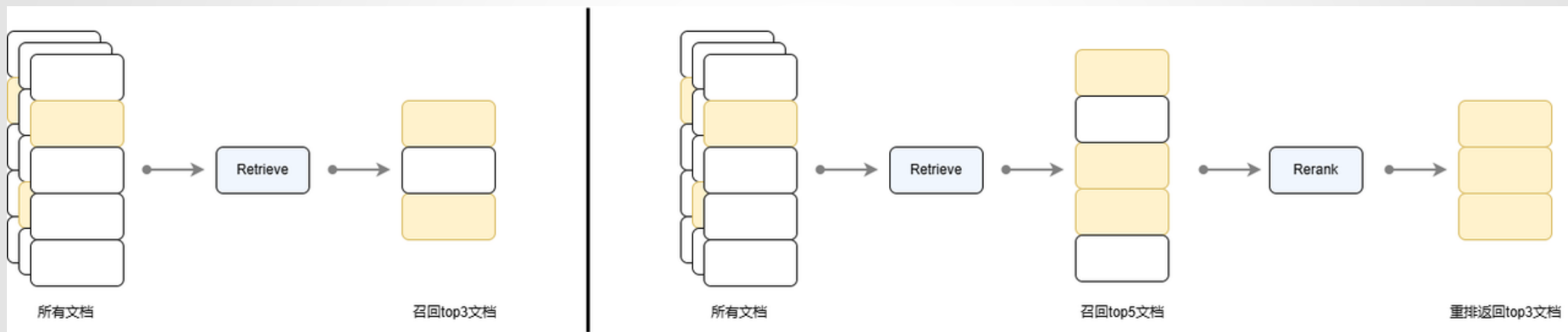
- 其中，|D| 为文档长度，avgdl 为平均长度，k1 和 b 是调节参数，常设 $k_1 \in [1.2, 2.0]$ ， $b=0.75$ 。k1 控制词频权重，越大词频影响越强；b 控制长度归一化，b 越大，文档长度影响越明显。

总结：这三种相似度计算方式（内积、余弦、BM25）各有优势，应根据向量类型和任务需求灵活选择。总结了稀疏与稠密向量常用的相似度组合。

| | 传统方法 (TF-IDF) | 基于BERT的方法 (BGE-M3) |
|------|---------------|--------------------|
| 稀疏检索 | bm25 | IP |
| 稠密检索 | cosine | cosine |

优化检索策略 – 重排序

召回重排是常见的两阶段检索策略，先用检索器初筛文档，再用更强模型重排序提升准确率。由于检索阶段可能丢失语义细节，仅靠相似度分数难以保证准确性，而直接用排序模型处理全量文档又开销过大，因此先召回后重排能在效率与效果之间取得平衡。



上图展示了两阶段检索中 Retriever 和 Reranker 的分工：

- Retriever 负责快速从海量文档中筛选相关候选，常用语义检索或 BM25 方法，但可能出现信息丢失或语义不足。
- 为提高准确性，Reranker 通过更强大的排序模型（如 bge-reranker-base）对候选文档进行精细的语义分析和排序，提升检索精度。整体流程实现了“先快后精”的漏斗结构，兼顾效率与效果。

整体流程实现了 **“先快后精”** 的漏斗结构，兼顾效率与效果。



目录



1. 上节回顾



2. RAG效果评测



3. 提升RAG系统的召回率



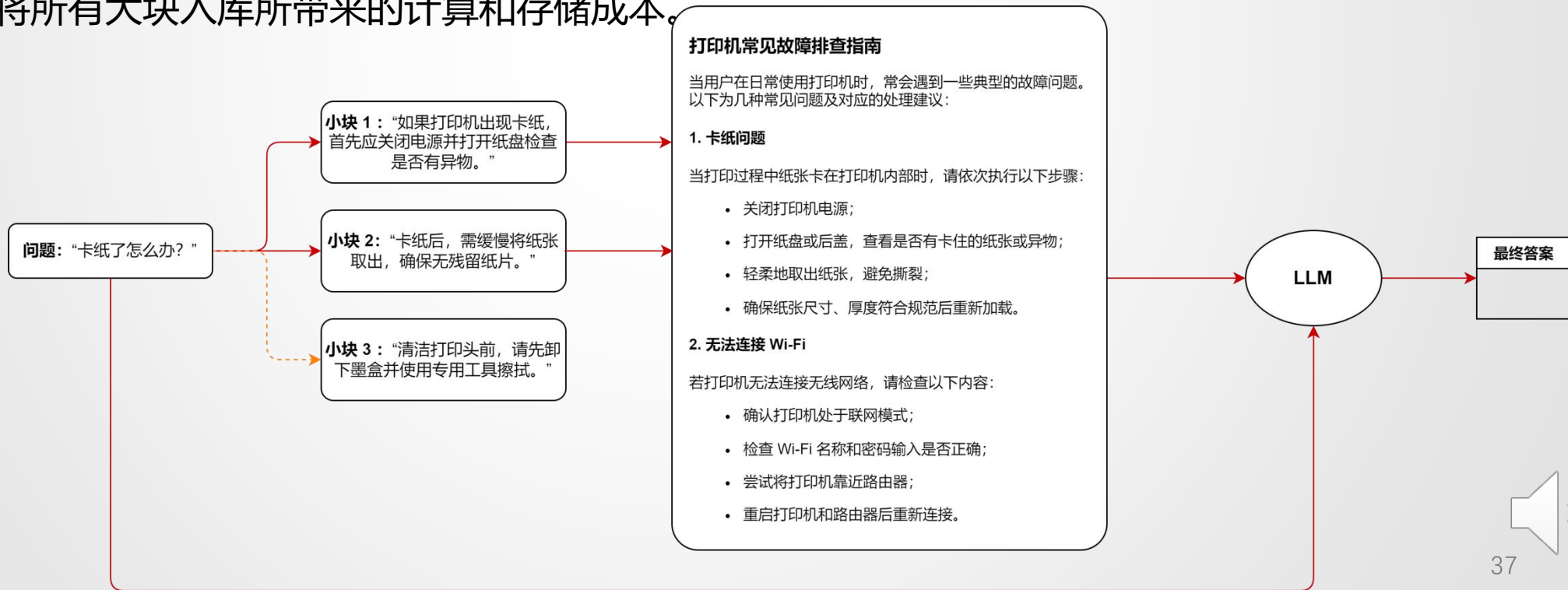
4. 多路召回的RAG



大小块策略

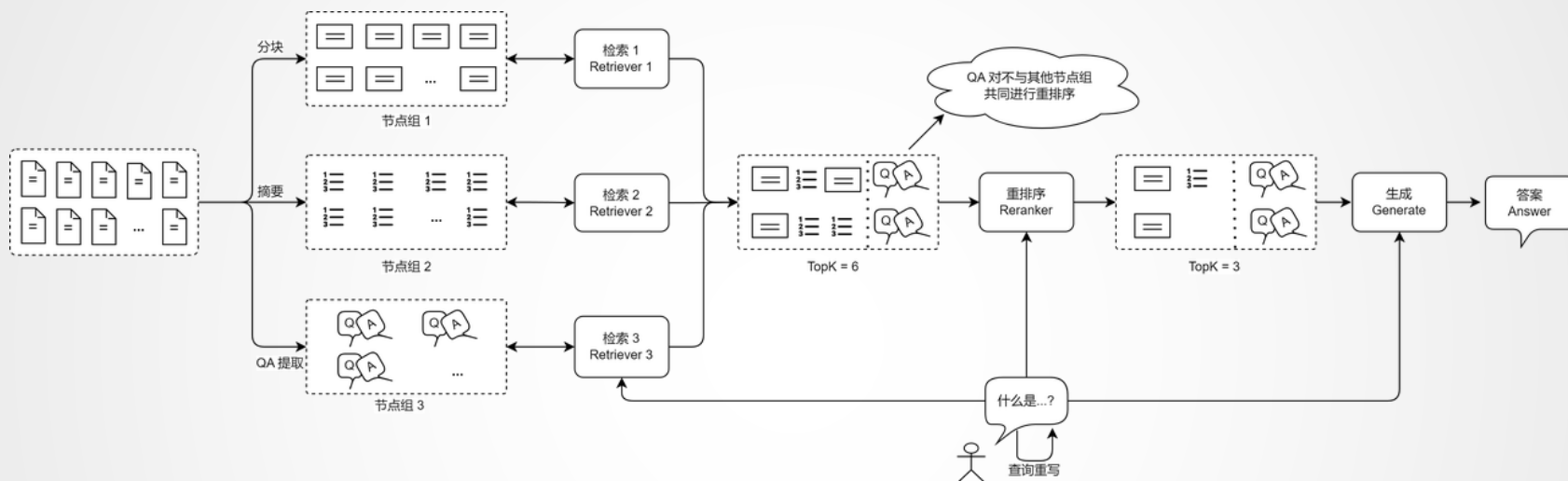
在 RAG 系统中，“**小块召回、大块生成**” 是一种常见且高效的策略，其核心思想是：

1. **小块用于高精度召回文档先按较小粒度进行切分（如按段落、句子）**，每个小块携带更精细的语义信息，便于精准计算与用户查询的相似度。检索阶段只对小块向量进行相似度比对，从而更容易找到与用户查询语义接近的内容。
2. **大块用于上下文生成一旦某个小块被成功召回，系统会找到该小块所属的大块**（如一整节内容或完整子话题），将其**作为上下文提供给大模型**，用于生成更完整、更连贯的回答。这样做的好处是：避免小块信息太碎，大模型理解困难；同时减少将所有大块入库所带来的计算和存储成本。



多路召回RAG

在 RAG 系统中，多路召回融合多种检索方式提升召回率，结合重排序策略提高生成质量。通过多个检索器处理不同来源或粒度的信息，再融合排序，能构建更完整上下文，优化整体效果。优化流程包括：



- 1. 不同节点组检索组合：**多个检索器并行查询不同数据，如原文块、摘要或问答对，提升召回率。
- 2. 不同表示策略组合：**结合原文检索与向量检索，或者使用多种不同的向量检索提升匹配效果。
- 3. 不同相似度计算方式组合：**综合使用dot, bm25, cosine等多种相似度计算策略进行效果提升
- 4. 综合运用上述组合：**综合上述1,2,3中的不同策略，进行组合进行效果提升

💡总结：相较传统单一检索，多路召回 RAG 能处理复杂查询、覆盖更广内容，并增强答案的完整性与准确性，尤其适用于领域或信息密集型任务。

在评测集上实际尝试各种策略单独使用，找到**优势互补**的策略

策略1:

- ✓ 问题1
- ✓ 问题2
- ✓ 问题3
- ✗ 问题4
- ✗ 问题5
- ✗ 问题6

策略2:

- ✓ 问题1
- ✓ 问题2
- ✓ 问题3
- ✗ 问题4
- ✗ 问题5
- ✗ 问题6

策略3:

- ✓ 问题1
- ✗ 问题2
- ✓ 问题3
- ✗ 问题4
- ✓ 问题5
- ✗ 问题6

策略4:

- ✗ 问题1
- ✗ 问题2
- ✗ 问题3
- ✓ 问题4
- ✓ 问题5
- ✓ 问题6

Q&A

1. RAG召回的时候，有的问题关联的文档较多，有的问题较少，该如何设置topk?
2. 垂直领域场景是否需要大模型训练?



感谢聆听
Thanks for Listening

