

# [第4周] 检索增强生成

## ETM15：5秒内向我解释

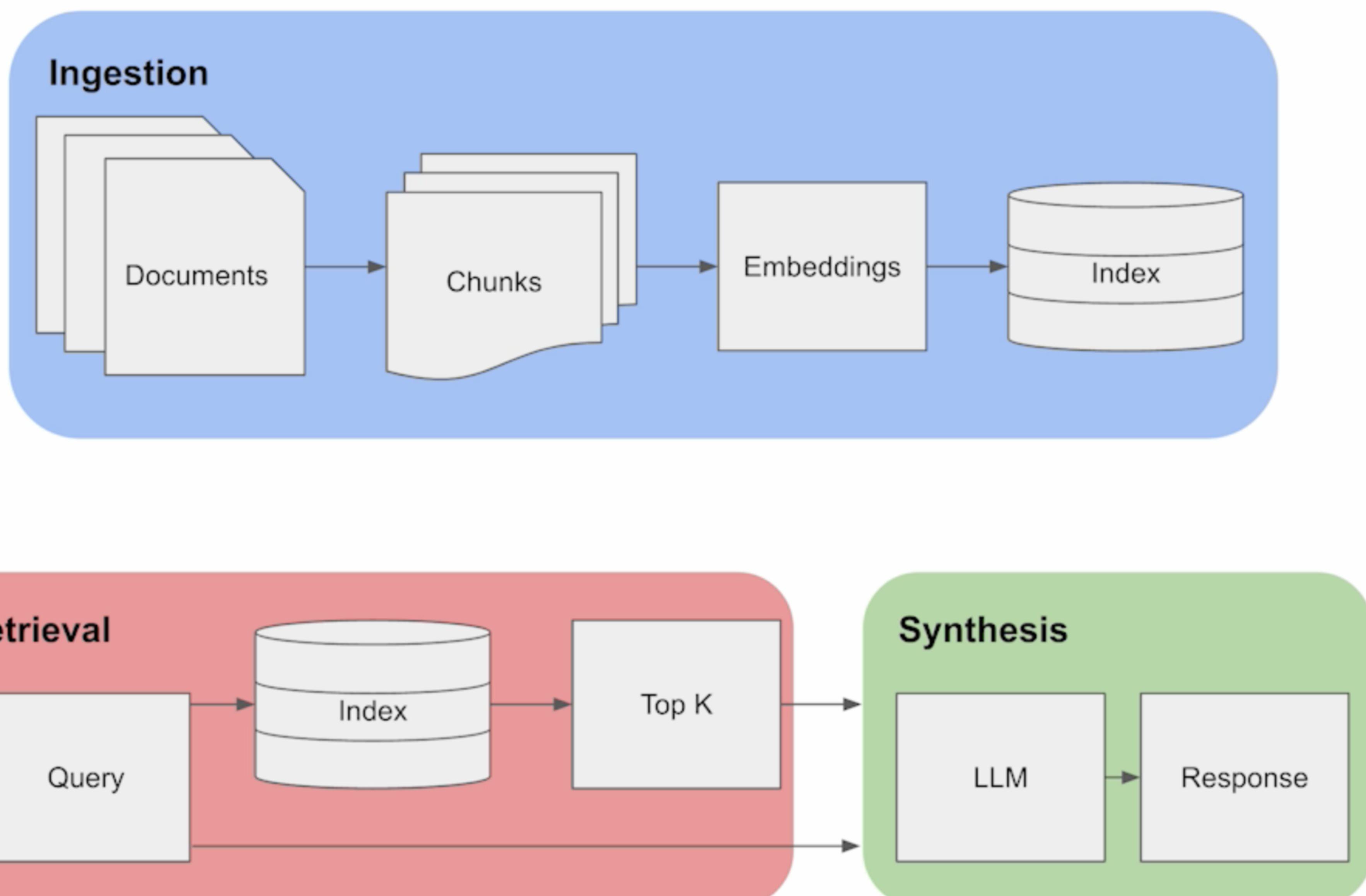
在本周的内容中，我们将深入探讨检索增强生成 (RAG)，这是一种 AI 框架，通过在响应生成过程中集成来自外部来源的实时上下文相关信息来增强大型语言模型的功能。它解决了 LLM 的局限性，例如不一致和缺乏领域特定知识，从而降低了生成错误或幻觉响应的风险。

RAG 分为三个关键阶段：提取、检索和综合。在提取阶段，文档被分割成更小、更易于管理的块，然后将其转换为嵌入并存储在索引中，以便高效检索。检索阶段涉及利用索引在收到用户查询时根据相似性指标检索前  $k$  个相关文档。最后，在综合阶段，LLM 利用检索到的信息及其内部训练数据来制定对用户查询的准确响应。

我们将讨论 RAG 的历史，然后深入研究 RAG 的关键组成部分，包括提取、检索和合成，并详细了解每个阶段的流程和改进策略。我们还将讨论与 RAG 相关的各种挑战，例如数据提取复杂性、高效嵌入和泛化微调，并针对每个挑战提出解决方案。

## 什么是 RAG？（回顾）

检索增强生成 (RAG) 是一种 AI 框架，通过在生成过程中整合来自外部来源的最新且上下文相关的信息来提高 LLM 生成的响应质量。它解决了 LLM 中不一致和缺乏领域特定知识的问题，从而降低了出现幻觉或错误响应的可能性。RAG 涉及两个阶段：检索（搜索和检索相关信息）和内容生成（LLM 根据检索到的信息及其内部训练数据合成答案）。这种方法提高了准确性，允许源验证，并减少了对持续模型再训练的需求。



图片来源：<https://www.deeplearning.ai/short-courses/langchain-for-lm-application-development/>

上图概述了基本的 RAG 管道，由三个关键组件组成：

### 1. 摄入：

- 文档被分割成块，然后从这些块生成嵌入，随后存储在索引中。
- 块对于根据给定的查询精确定位相关信息至关重要，类似于标准检索方法。

### 2. 检索：

- 利用嵌入索引，系统在收到查询时根据嵌入的相似性检索前  $k$  个文档。

### 3. 合成：

- 通过将块作为上下文信息进行检查，LLM 利用这些知识来制定准确的响应。

💡 与以前的领域自适应方法不同，需要强调的是，RAG 根本不需要任何模型训练。当提供特定领域数据时，它可以轻松应用，而无需训练。

## 历史

RAG，即检索增强生成，首次出现在Meta 的[这篇](#)论文中。这个想法是为了应对大型预训练语言模型在有效获取和操纵知识方面的局限性而提出的。

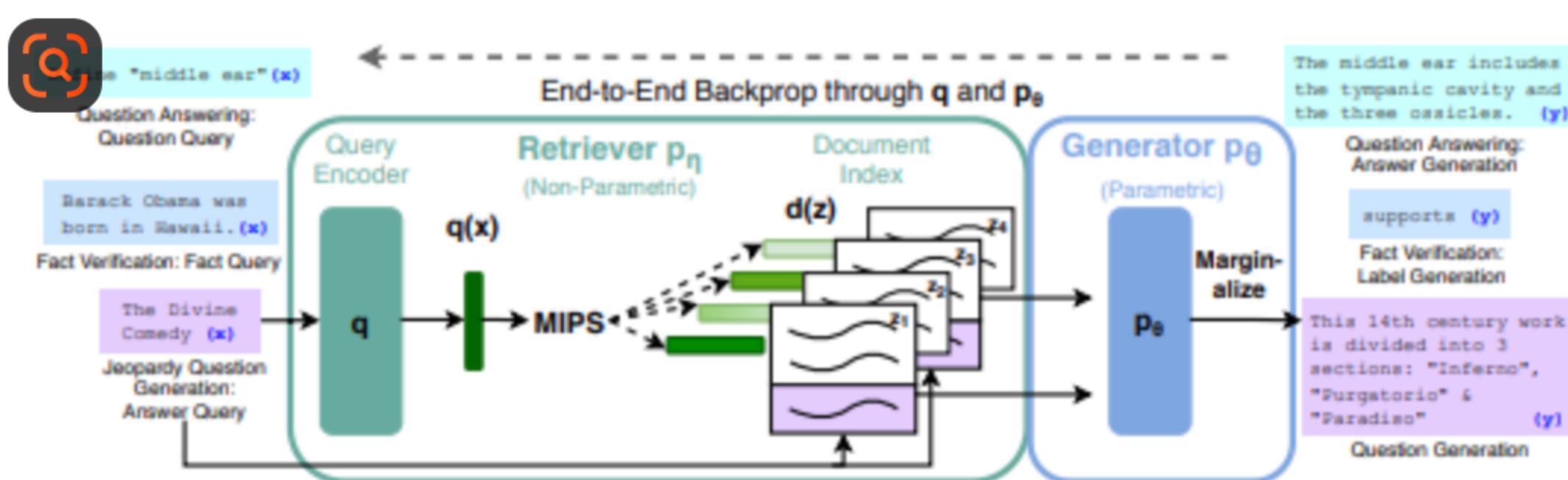


Image Source: [https://arxiv.org/pdf/2005.11401.pdf] (https://arxiv.org/pdf/2005.11401.pdf)



以下是作者如何介绍问题并提供解决方案的简短摘要：

RAG 的出现是因为，尽管大型语言模型擅长记忆事实和执行特定任务，但它们在精确使用和操纵这些知识方面却举步维艰。这在知识密集型任务中表现得很明显，其他专门的模型表现优于它们。作者指出了现有模型面临的挑战，例如难以解释决策和跟上现实世界的变化。在 RAG 之前，混合参数和非参数记忆的混合模型取得了令人鼓舞的结果。REALM 和 ORQA 等示例将掩蔽语言模型与检索器相结合，显示出了这方面的积极成果。

随后，RAG 应运而生，它以灵活的微调方法改变了游戏规则，可用于检索增强生成。RAG 将预训练的参数记忆（如 seq2seq 模型）与来自 Wikipedia 密集向量索引的非参数记忆相结合，通过预训练的神经检索器（如密集段落检索器 (DPR)）访问。RAG 模型旨在通过微调将预训练的参数记忆生成模型与非参数记忆相结合，从而增强预训练的参数记忆生成模型。RAG 中的 seq2seq 模型使用神经检索器检索到的潜在文档，从而创建了一个端到端训练的模型。训练涉及对任何 seq2seq 任务进行微调，同时学习生成器和检索器。然后使用每个输出或每个标记的 top-K 近似值处理潜在文档。

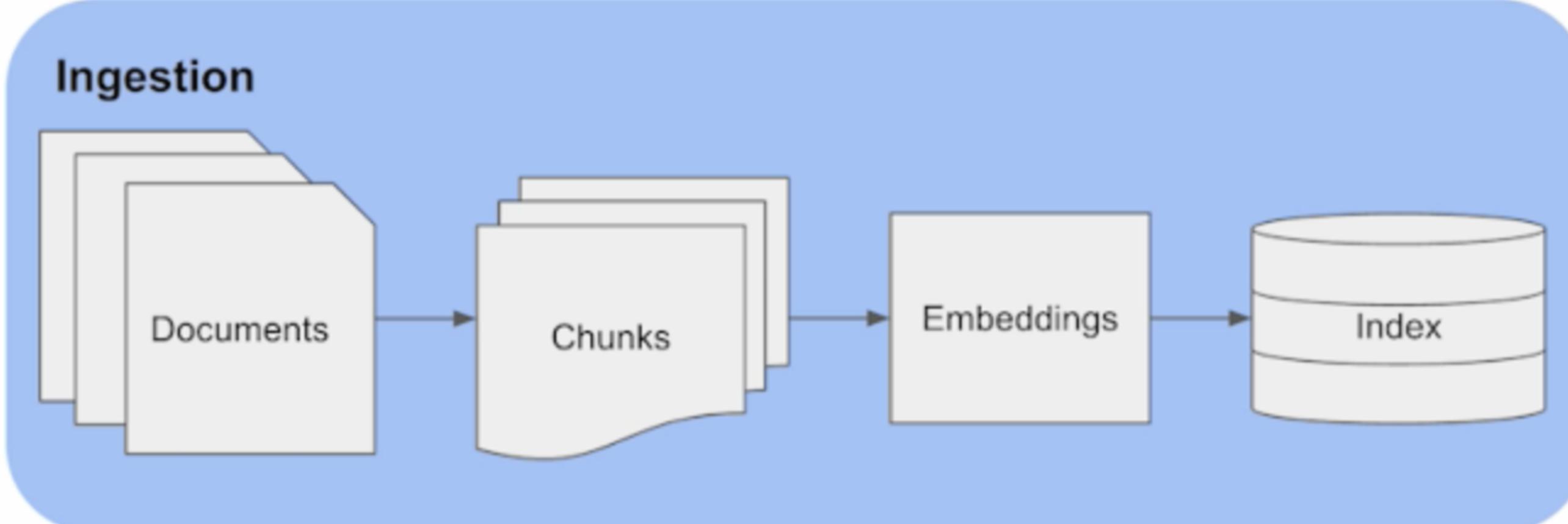
RAG 的主要意义在于摆脱了过去提出在系统中添加非参数记忆的方法。相反，RAG 探索了一种新方法，其中参数和非参数记忆组件都经过预先训练并填充了大量知识。在实验中，RAG 通过在开放域问答中取得一流的结果并在事实验证和知识密集型生成方面超越以前的模型证明了其价值。RAG 的另一个胜利是表明它可以适应，允许非参数记忆被换出和更新，以保持模型知识在不断变化的世界中保持新鲜。

## 关键组件

如前所述，RAG 的关键要素涉及摄取、检索和合成过程。现在，让我们深入研究每个组件。

### 摄入

在 RAG 中，摄取过程是指在模型利用数据生成响应之前对其进行处理和准备。

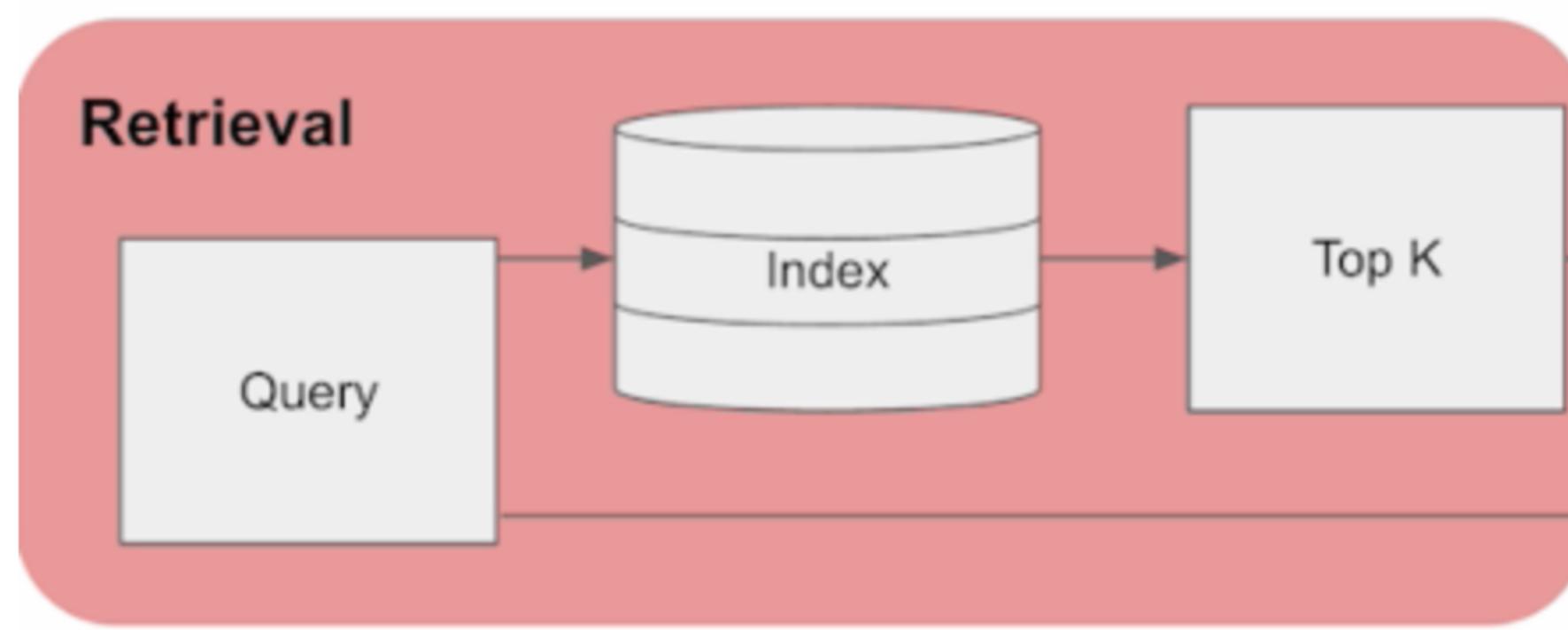


此过程涉及 3 个关键步骤：

- 分块：**将输入文本分解为更小、更容易管理的片段或块。这可以基于文本的大小、句子或其他自然划分。我们将在下一节中深入探讨分块策略。例如，考虑一篇关于文艺复兴的综合文章。分块过程包括根据自然断点将文章分解为可管理的片段，例如段落或不同的历史时期（例如，早期文艺复兴、盛期文艺复兴）。这些片段中的每一个都成为一个块，使语言模型能够进行重点分析。
- 嵌入：**将文本或块转换为矢量格式，以计算友好的方式捕捉基本特征。此步骤对于语言模型的高效处理至关重要。从上例中可以看出，一旦识别出文章片段，嵌入过程就会将每个块的内容转换为矢量格式。例如，关于文艺复兴盛期的部分可以嵌入到捕捉关键艺术、文化和历史方面的矢量中。这种矢量表示增强了模型理解和处理块内细微信息的能力。
- 索引：**以优化的结构化格式组织嵌入数据，以便快速高效地检索。这通常涉及为每个文档创建一个向量表示，并以可搜索的格式存储这些向量，例如向量数据库或搜索引擎。在我们讨论的示例中，索引数据库是通过组织这些历史事件的向量表示来创建的。每个块（现在表示为一个向量）都已编入索引，以便高效检索。当用户查询文艺复兴的某个特定方面时，索引可以快速识别和检索最相关的块，从而提供丰富的上下文响应。

### 检索

检索组件涉及以下步骤：

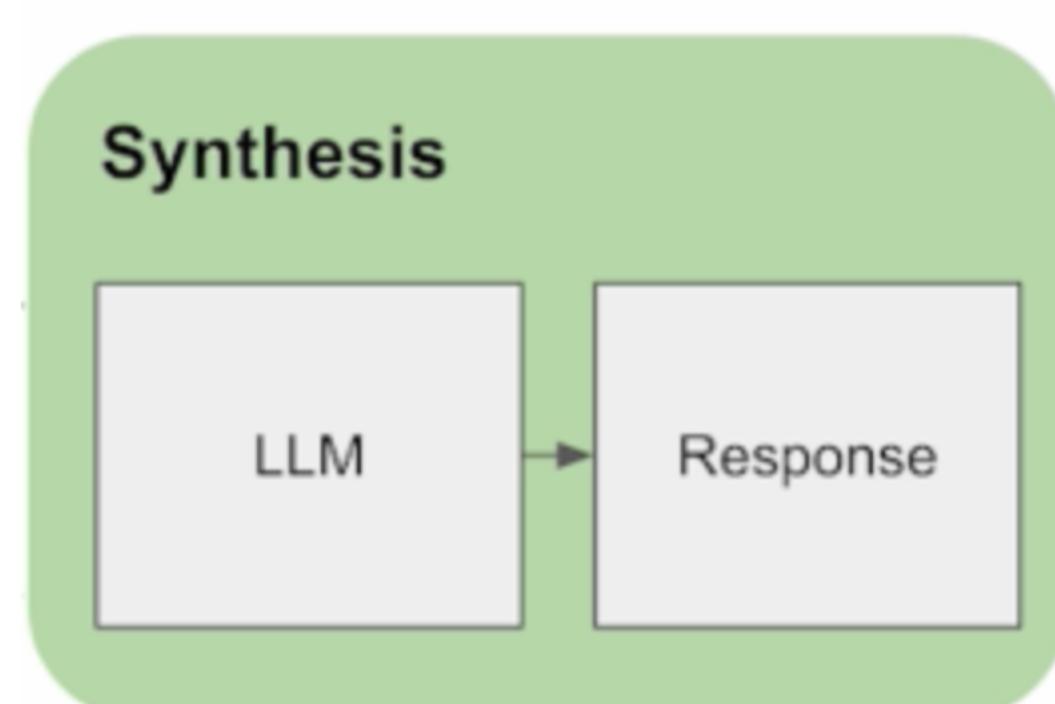


- 用户查询:** 用户向 LLM 提出自然语言查询。例如，假设我们已经按照上述方法完成了文艺复兴时期文章的提取过程，并且用户提出了一个查询，“告诉我关于文艺复兴时期的事情。”
- 查询转换:** 查询被发送到嵌入模型，该模型将自然语言查询转换为数字格式，从而创建嵌入或向量表示。嵌入模型与在提取阶段用于嵌入文章的模型相同。
- 向量比较:** 将查询的数字向量与上一阶段创建的知识库索引中的向量进行比较。这涉及测量查询向量与索引中存储的向量之间的相似度或距离度量（通常是余弦相似度）。
- Top-K 检索:** 然后，系统从知识库中检索与查询向量相似度最高的前 K 个文档或段落。此步骤涉及根据向量相似度选择预定义数量 (K) 的最相关文档。这些嵌入可能包含有关文艺复兴不同方面的信息。
- 数据检索:** 系统从知识库中选定的前 K 个文档中检索实际内容或数据。这些内容通常采用人类可读的形式，表示与用户查询相关的信息。

因此，在检索阶段结束时，LLM 可以访问与用户查询最相关的知识库部分的相关上下文。在此示例中，检索过程可确保用户收到有关文艺复兴的充分信息，并利用知识库中存储的历史文档提供丰富的上下文信息。

## 合成

综合阶段与常规 LLM 生成非常相似，不同之处在于 LLM 现在可以从知识库中访问其他上下文。LLM 将其自己的语言生成与从知识库中检索到的信息相结合，向用户呈现最终答案。响应可能包括对特定文档或历史来源的引用。



## RAG 挑战

尽管 RAG 似乎是将 LLM 与知识相结合的一种非常直接的方法，但 RAG 仍然存在下面提到的开放研究和应用挑战。

- 数据采集复杂性:** 处理采集大量知识库的复杂性需要克服工程挑战。例如，有效地并行化请求、管理重试机制和扩展基础设施都是关键考虑因素。想象一下采集大量不同的数据源（例如科学文章），并确保高效处理后续检索和生成任务。
- 高效嵌入:** 确保高效嵌入大型数据集带来了诸多挑战，例如解决速率限制、实现强大的重试逻辑以及管理自托管模型。考虑这样一个场景：AI 系统需要嵌入大量新闻文章，这需要制定策略来处理不断变化的数据、同步机制并优化嵌入成本。
- 矢量数据库注意事项:** 将数据存储在矢量数据库中需要考虑诸多事项，例如了解计算资源、监控、分片和解决潜在瓶颈。考虑一下维护用于各种文档的矢量数据库所涉及的挑战，每个文档的复杂性和重要性程度各不相同。
- 微调和泛化:** 针对特定任务对 RAG 模型进行微调，同时确保其在各种知识密集型 NLP 任务中的泛化，这极具挑战性。例如，与涉及创造性语言生成的任务相比，在问答任务中实现最佳性能可能需要不同的微调方法，需要谨慎平衡。
- 混合参数和非参数记忆:** 在 RAG 等模型中集成参数和非参数记忆组件会带来与知识修订、可解释性和避免幻觉相关的挑战。考虑一下确保语言模型将其预训练知识与动态检索的信息相结合、避免不准确并保持一致性的难度。
- 知识更新机制:** 随着现实世界知识的发展，开发更新非参数记忆的机制至关重要。想象一下这样一个场景：RAG 模型需要适应医学等领域不断变化的信息，因为医学领域不断涌现新的研究成果和治疗方法，需要及时更新才能做出准确的反应。

## 改进 RAG 组件（提取）

### 1.更好的分块策略

在增强 RAG 组件的提取过程的背景下，采用高级分块策略对于高效处理文本数据是必要的。在简单的 RAG 管道中，采用固定策略，即固定数量的单词或字符形成单个块。

考虑到大型数据集的复杂性，最近正在使用以下策略：

- 基于内容的分块:** 使用词性标记或句法分析等技术根据含义和句子结构对文本进行分解。这可以保留文本的意义和连贯性。但是，这种分块的一个考虑因素是它需要额外的计算资源和算法复杂性。
- 句子分块:** 使用句子边界识别或语音片段将文本分解为完整且语法正确的句子。保持文本的统一性和完整性，但可能生成大小不一、缺乏同质性的块。

3. 递归分块：将文本拆分为不同级别的块，从而创建分层且灵活的结构。提供更大的文本粒度和多样性，但管理和索引这些块会增加复杂性。

## 2.更好的索引策略

改进的索引功能可实现更高效的信息搜索和检索。当数据块被正确索引时，可以更轻松地快速定位和检索特定信息。一些改进的策略包括：

1. **详细索引**：按子部分（例如句子）进行分块，并根据其位置为每个块分配一个标识符，并根据内容分配一个特征向量。提供特定的上下文和准确性，但需要更多的内存和处理时间。
2. **基于问题的索引**：通过知识域（例如主题）对数据块进行分组，并根据其类别为每个数据块分配一个标识符，并根据相关性为每个数据块分配一个特征向量。直接与用户请求保持一致，可提高效率，但可能会导致信息丢失和准确性降低。
3. **使用块摘要优化索引**：使用提取或压缩技术为每个块生成摘要。根据摘要分配标识符，并根据相似性分配特征向量。提供更大的综合性和多样性，但在生成和比较摘要时需要复杂性。

## 改进 RAG 组件（检索）

### 1.假设性问题和HyDE：

引入假设性问题涉及为每个块生成一个问题，将这些问题嵌入向量中，并针对此问题向量索引执行查询搜索。由于查询和假设性问题之间的语义相似度高于实际块，因此这提高了搜索质量。相反，HyDE（假设性响应提取）涉及根据查询生成假设性响应，通过利用查询的向量表示及其假设性响应来提高搜索质量。

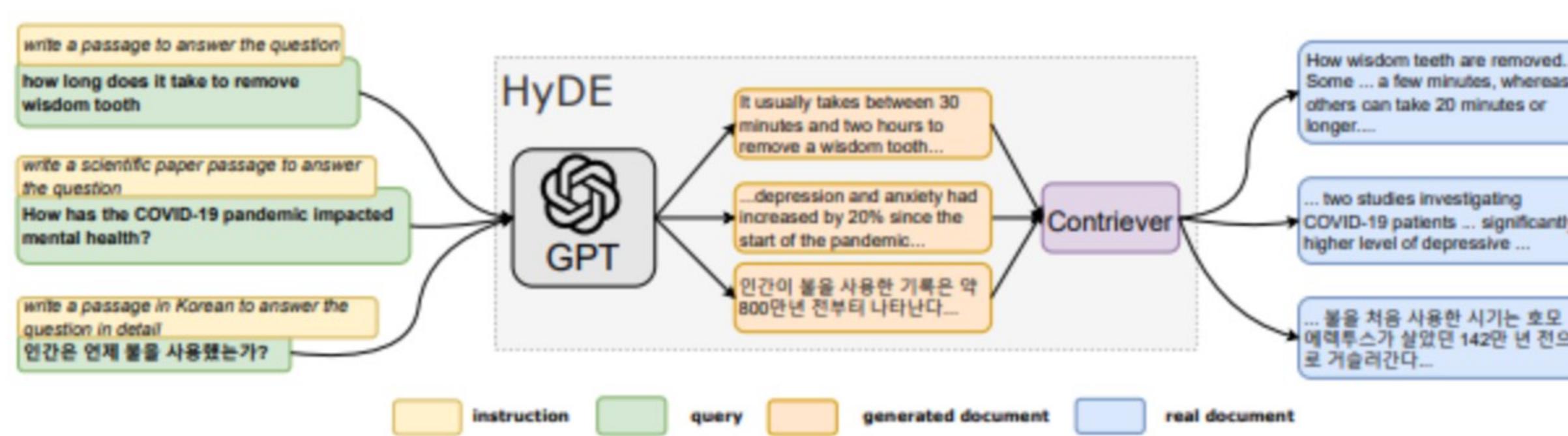
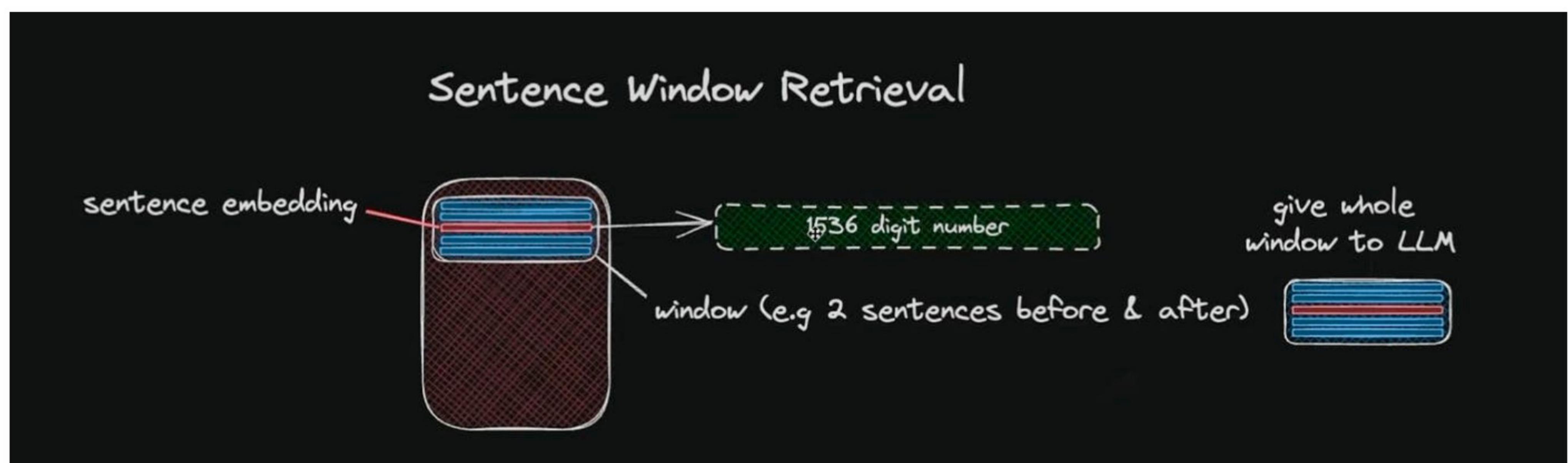


Image Source: [https://arxiv.org/pdf/2212.10496.pdf](https://arxiv.org/pdf/2212.10496.pdf)

### 2.上下文丰富：

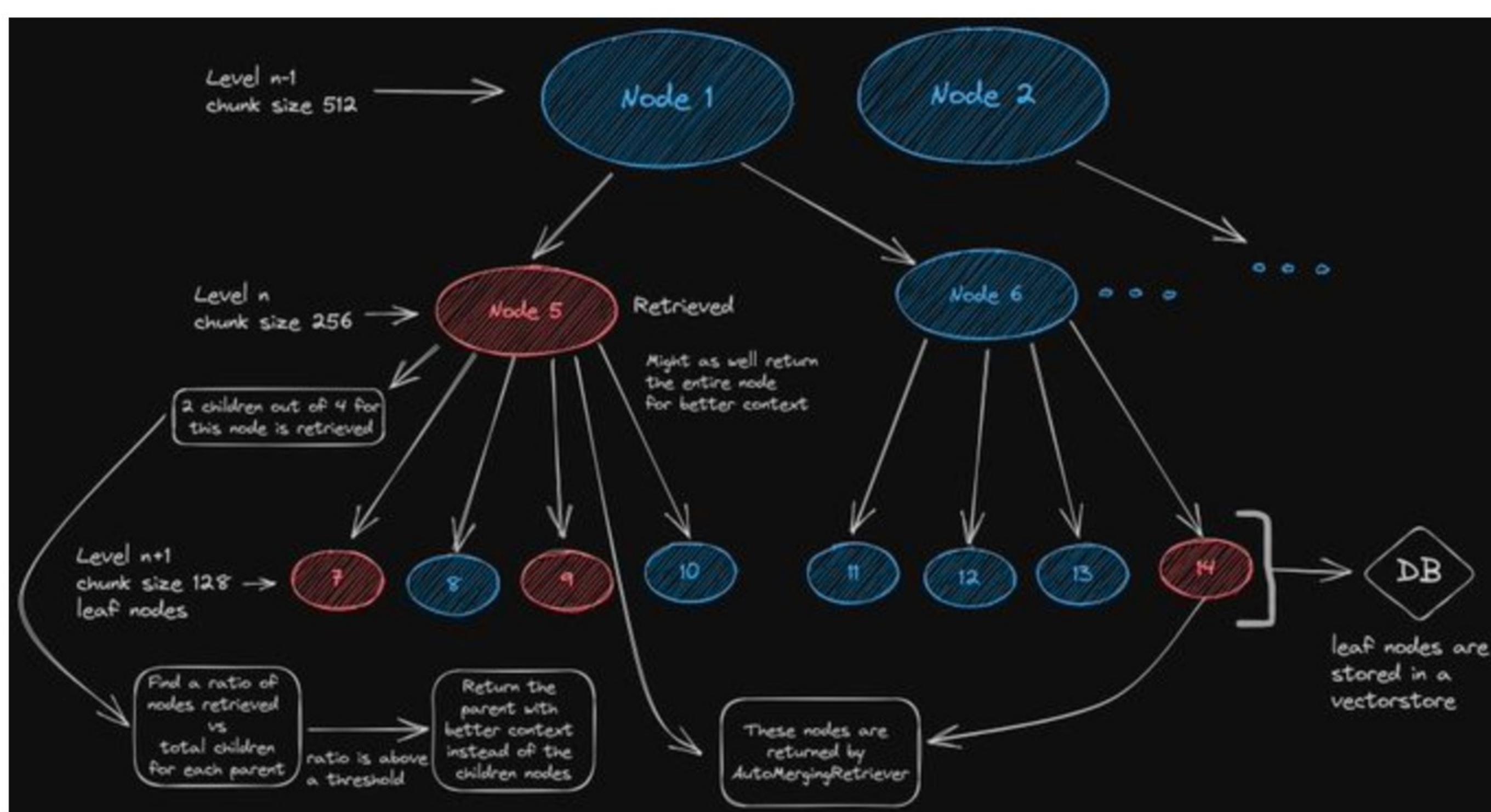
此处的策略旨在检索更小的块以提高搜索质量，同时结合周围环境供语言模型进行推理。可以探索两种选择：

1. **句子窗口检索**：将每个句子分别嵌入文档中，以实现查询和上下文之间余弦距离搜索的高精度。检索到最相关的单个句子后，通过在检索到的句子之前和之后包含指定数量的句子来扩展上下文窗口。然后将扩展的上下文发送到 LLM，以根据提供的查询进行推理。目标是增强 LLM 对检索到的句子周围上下文的理解，从而实现更明智的响应。



图片来源：https://medium.com/@shivansh.kaushik/advanced-text-retrieval-with-elasticsearch-llamaindex-sentence-window-retrieval-cb5ea720aa44

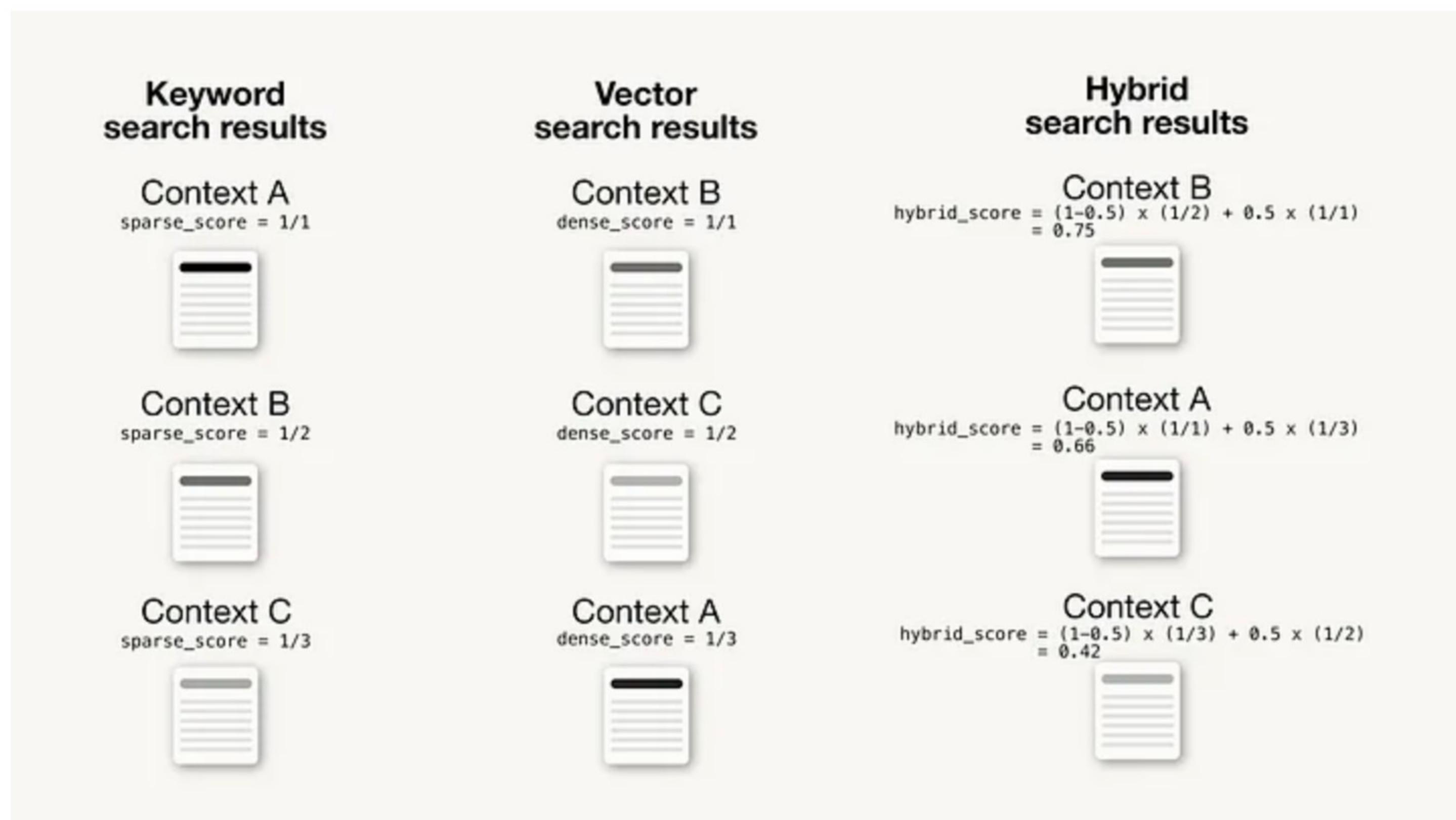
1. **自动合并检索器**：在这种方法中，文档最初被分成较小的子块，每个子块引用较大的父块。在检索过程中，首先获取较小的块。如果在检索到的顶部块中，超过指定数量的块链接到同一个父节点（较大的块），则输入到 LLM 的上下文将被此父节点替换。这个过程可以比作自动将几个检索到的块合并为一个较大的父块，因此得名“自动合并检索器”。该方法旨在捕获粒度和上下文，从而有助于从 LLM 获得更全面、更连贯的响应。



图片来源：<https://twitter.com/clusteredbytes>

### 3.融合检索或混合搜索：

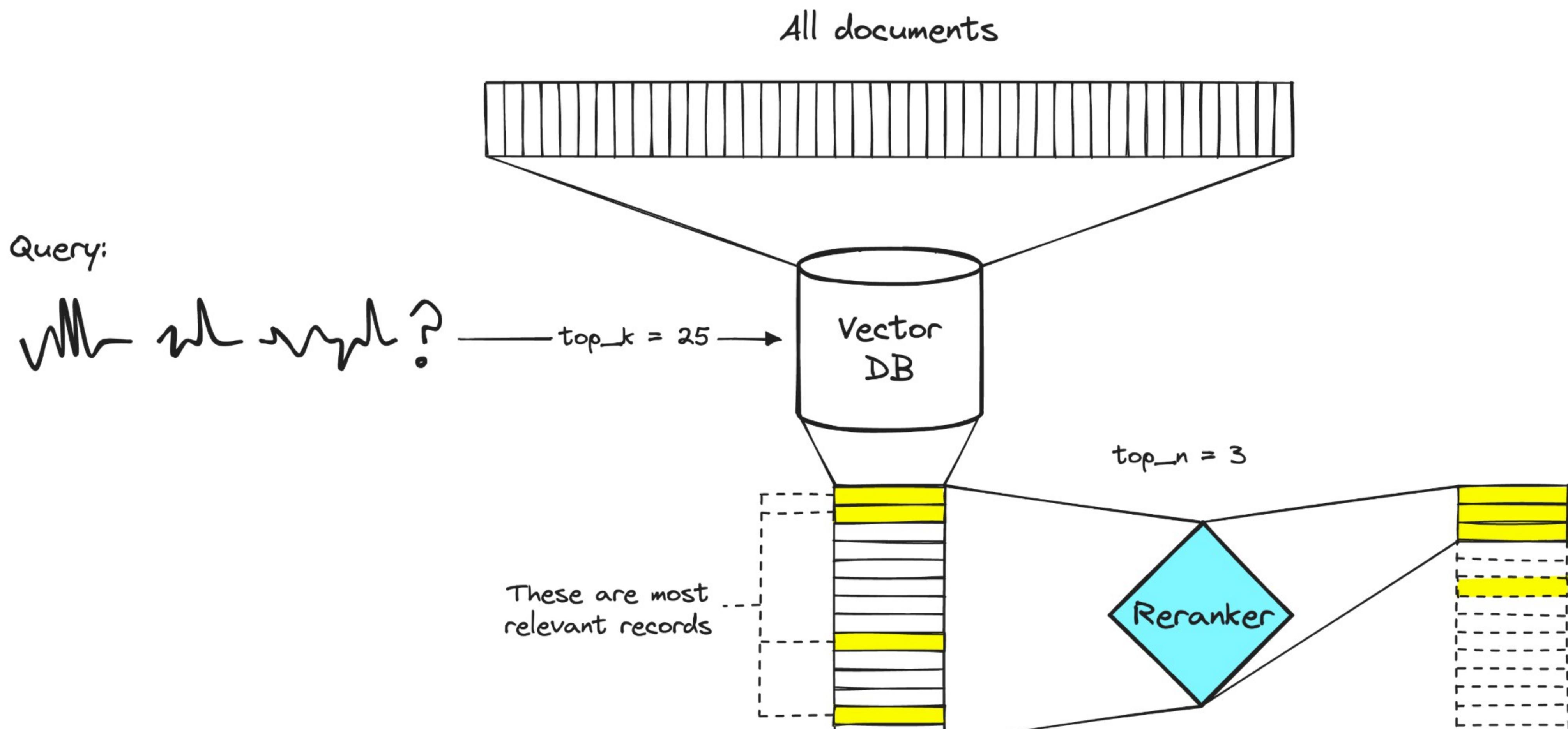
该策略将传统的基于关键词的搜索方法与当代语义搜索技术相结合。通过将 tf-idf (词频-逆文档频率) 或 BM25 等各种算法与基于向量的搜索相结合，RAG 系统可以充分利用语义相关性和关键词匹配的优势，从而获得更全面、更全面的搜索结果。



图片来源：<https://towardsdatascience.com/improving-retrieval-performance-in-rag-pipelines-with-hybrid-search-c75203c2f2f5>

### 4.重新排序和过滤：

检索后细化是通过过滤、重新排序或转换来执行的。LlamaIndex 提供各种后处理器，允许根据相似度得分、关键字、元数据过滤结果，或使用 LLM 或句子转换器交叉编码器等模型进行重新排序。此步骤先于将检索到的上下文最终呈现给 LLM 以生成答案。



图片来源: <https://www.pinecone.io/learn/series/rag/rerankers/>

## 4. 查询转换和路由 [来源]

查询转换方法通过将复杂查询分解为子问题（扩展）并通过重写改进措辞不当的查询来增强检索。而动态查询路由可优化不同来源的数据检索。以下是流行的方法。

### 查询转换

1. **查询扩展**: \*查询扩展将输入分解为子问题，每个子问题都是一个更狭窄的检索挑战。例如，一个关于物理的问题可以分解为一个关于用户查询背后的物理原理的问题（以及 LLM 生成的答案）。
2. **查询重写**: 针对结构不当或措辞不当的用户查询，[重写-检索-阅读](#)方法涉及重新措辞问题以提高检索效率。本文详细解释了该方法。
3. **查询压缩**: 在用户问题出现在更广泛的聊天对话之后的情况下，可能需要完整的对话上下文才能回答该问题。查询压缩用于将聊天记录压缩为最终问题以供检索。

### 查询路由

1. **动态查询路由**: 数据驻留在何处的问题在 RAG 中至关重要，尤其是在具有各种数据存储的生产环境中。由 LLM 支持的动态查询路由可有效地将传入查询定向到适当的数据存储。此动态路由可适应不同的来源并优化检索过程。

## 改进 RAG 组件（生成）

生成 LLM 的最直接方法是将所有相关上下文片段连接起来，超过预定义的相关性阈值，并将它们与查询一起在单个实例中呈现给 LLM。但是，存在更高级的替代方案，需要多次调用 LLM 来迭代增强检索到的上下文，最终生成更精致、更完善的答案。下面说明了一些方法。

### 1. 响应合成方法:

涉及 3 个步骤

1. **迭代细化**: 通过逐块发送检索到的上下文到语言模型来细化答案。
2. **总结**: 总结检索到的上下文以适应提示并生成简洁的答案。
3. **多重答案和连接**: 根据不同的上下文块生成多个答案，然后连接或总结它们。

### 2. 编码器和 LLM 微调:

这种方法涉及在我们的 RAG 管道内对 LLM 模型进行微调。

1. **编码器微调**: 微调 Transformer 编码器以获得更好的嵌入质量和上下文检索。
2. **排名器微调**: 使用交叉编码器对检索到的结果进行重新排名，特别是在对基础编码器缺乏信任的情况下。
3. **RA-DIT 技术**: 使用 RA-DIT 之类的技术根据查询、上下文和答案的三元组调整 LLM 和检索器。

## 阅读/观看这些资源（可选）

1. 构建可用于生产的 RAG 应用程序: <https://www.youtube.com/watch?v=TRjq7t2Ms5I>
2. 亚马逊关于 RAG 的文章 - <https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-foundation-models-customize-rag.html>
3. Huggingface RAG 工具 - [https://huggingface.co/docs/transformers/model\\_doc/rag](https://huggingface.co/docs/transformers/model_doc/rag)
4. 12 个 RAG 痛点及建议解决方案 - <https://towardsdatascience.com/12-rag-pain-points-and-proposed-solutions-43709939a28c>

## 阅读这些论文（可选）

1. [大型语言模型的检索增强生成：一项调查](#)
2. [设计检索增强生成系统时的七个失败点](#)