一文看懂什么是RAG（检索增强生成）

Posted 2024-04-6 • Updated 2024-04- 23 • 64~83 min read

> RAG（中文为检索增强生成） = 检索技术 + LLM 提示。例如，我们向 LLM 提问一个问题（answer），RAG 从各种数据源检索相关的信息，并将检索到的信息和问题（answer）注入到 LLM 提示中，LLM 最后给出答案。

背景

检索增强生成（Retrieval Augmented Generation），简称 RAG，已经成为当前最火热的LLM应用方案。

理解不难，就是通过自有垂域数据库检索相关信息，然后合并成为提示模板，给大模型生成漂亮的回答。

经历23年年初那一波大模型潮，想必大家对大模型的能力有了一定的了解，但是当我们将大模型应用于实际业务场景时会发现，通用的基础大模型基本无法满足我们的实际业务需求，主要有以下几方面原因：

- **知识的局限性**

模型自身的知识完全源于它的训练数据，而现有的主流大模型（ChatGPT、文心一言、通义千问...）的训练集基本都是构建于网络公开的数据，对于一些实时性的、非公开的或离线的数据是无法获取到的，这部分知识也就无从具备。

- **幻觉问题**

  所有的AI模型的底层原理都是基于数学概率，其模型输出实质上是一系列数值运算，大模型也不例外，所以它有时候会一本正经地胡说八道，尤其是在大模型自身不具备某一方面的知识或不擅长的场景。而这种幻觉问题的区分是比较困难的，因为它要求使用者自身具备相应领域的知识。

- **数据安全性**

  对于企业来说，数据安全至关重要，没有企业愿意承担数据泄露的风险，将自身的私域数据上传第三方平台进行训练。这也导致完全依赖通用大模型自身能力的应用方案不得不在数据安全和效果方面进行取舍。

而RAG是解决上述问题的一套有效方案。

什么是RAG?

RAG（Retrieval Augmented Generation）为生成式模型提供了与外部世界互动提供了一个很有前景的解决方案。

RAG的主要作用类似搜索引擎，找到用户提问最相关的知识或者是相关的对话历史，并结合原始提问（查询），创造信息丰富的prompt，指导模型生成准确输出。其本质上应用了情境学习（In-Context Learning）的原理。

说直接点：

> RAG（检索增强生成） = 检索技术 + LLM 提示

RAG的特点

从RAG的运作模式看，它具有以下几大特点：

- RAG依赖大语言模型来强化信息检索和输出，单独使用，能力受限。
- RAG能与外部数据无缝集成，较好地解决通用大模型在垂直、专业领域的知识短板。
- 一般情况下，RAG对接的私有数据库不参与大模型数据集训练，能在改善模型性能的同时，更好地保证数据隐私和安全。

- 同样是RAG，展现的效果并不统一，很大程度上受模型性能、外挂数据质量、AI算法、检索系统等多方面的影响。

RAG和大模型什么关系？

RAG和大模型关系：

1. RAG既是大语言模型（LLM）较为热门的应用开发架构，也是其在垂直领域的应用拓展。
2. 它能在LLM强大功能的基础上，通过拓展访问**特定领域数据库或内部知识库**，以补足通用模型在垂直领域的知识短板。
3. 在LLM的基础上，RAG能通过数据内循环，更好地解决数据来源问题。
4. 增强搜索与生成功能的同时，能减少行业用户对数据私密性与安全性的顾虑。这也推动其成为各类大模型落地项目不可缺少的技术组件。
5. RAG的利用，能有效减少模型幻觉的发生，进一步提升大模型检索和生成性能。

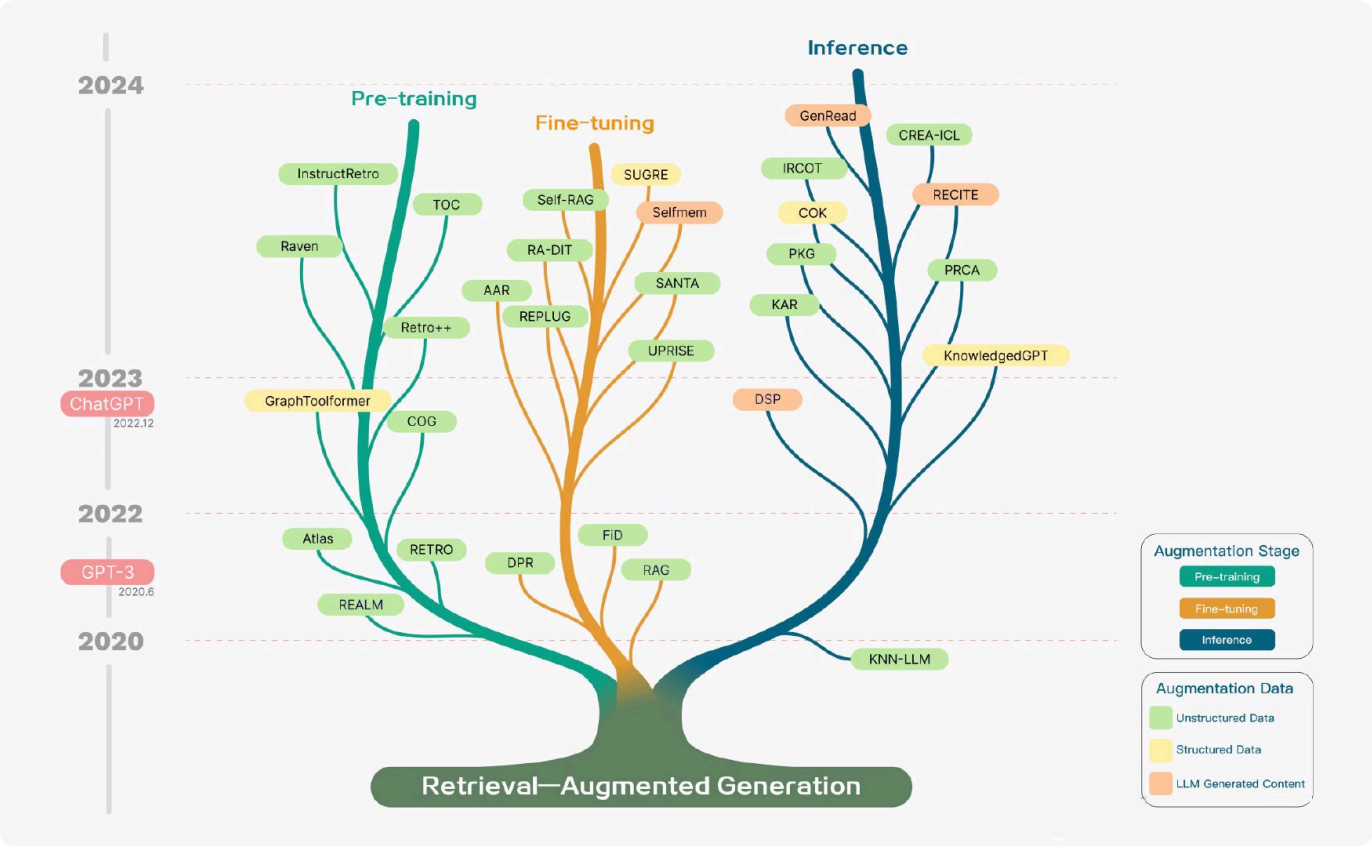
因为训练成本相对较低，RAG目前已拓展到企业信息库建设、AI文档问答、业务培训、科研等场景，搭配AI agent，极大地加快了大模型的商业化进程。

为了提高RAG的智能化程度和应用价值，人们甚至在原来的检索增强生成体系的基础上，推出了Self-RAG（即自反思的检索增强生成方法），以进一步提升检索效率和大模型生成质量。这算是RAG的一大升级与进化了。

RAG的发展阶段

RAG的概念是2020年提出的，第一次将预训练的检索器与预训练的seq2seq模型相结合，并进行端到端的微调。真正发扬光大是在2022年后，特别是ChatGPT的提出，NLP迈入大模型时代。

很多人不知道，RAG从诞生开始一直是研究预训练和SFT的，推理截断的RAG是在LLM时代后才开始井喷。因为高性能大模型的训练成本高，学术界和工业级试图通过推理阶段加入RAG模块，高性价比的方式整合外部知识来增强模型生成。RAG的检索范围也在逐步放开，早起的RAG重点关注开源的，非结构化知识；随着检索范围的扩大，高质量数据作为知识来源，逐步在减轻LLM大模型的错误知识和幻觉问题。最近也有很多关于知识图谱，自我检索等方向的研究，跨学科和LLM进行结合的探索。



RAG的初创与发展

RAG起源于2020年。通过结合预训练的检索器和生成器来捕获知识，提高了模型的可解释性和模块化。

大型语言模型时代的RAG发展：随着大型语言模型的兴起，RAG技术成为了提升聊天机器人和LLM实用性的关键工具。

RAG技术的演化：RAG通过优化检索器、生成器等关键部分，为大型模型中的复杂知识密集型任务提供了更高效的解决方案

从图1可以看出，RAG的增强阶段可以在pre-training预训练，Fine-tuning微调，Inference三个阶段；从增强的数据源，包括非结构化数据，结构化数据和LLM生成的内容三个途径。

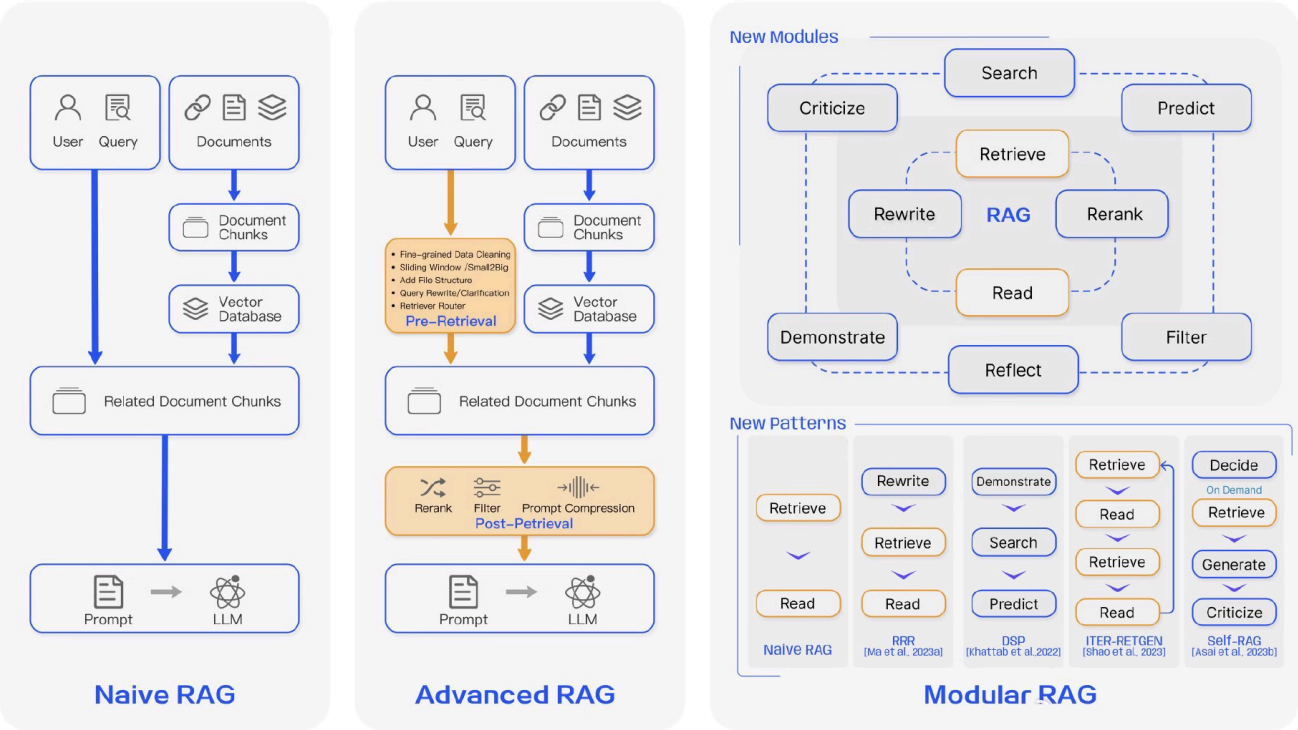
学术界和工业界，一般对RAG的定义有以下共识：

- **检索阶段**：利用编码模型根据问题检索相关文档。
- **生成阶段**：将检索到的上下文作为条件，系统生成文本。

RAG的研究范式

RAG研究范式不断发展演变。论文其划分为三类：**初级RAG、高级RAG和模块化RAG。**

从RAG的发展历史，可以总结三个阶段Naive RAG，Advanced RAG、Modular RAG。个人认为，**学术上的分类，是为了研究和归纳，将过去的方法总结和呈现。作为具有创新精神的我们，更多的是在业务上实践，在创新上去超越。**



初级RAG主要涉及“检索-阅读”过程。高级RAG使用更精细的数据处理，优化知识库索引，并引入多个或迭代检索。随着探索的深入，RAG整合了其他技术，如微调，导致模块化RAG范式的出现，该范式通过新模块丰富了RAG过程，并提供了更多的灵活性。

Naive RAG

经典的RAG过程，也被称为 Naive RAG。主要包括包括三个基本步骤：

1. 索引 — 将文档库分割成较短的 Chunk，并通过编码器构建向量索引。

2. 检索 — 根据问题和 chunks 的相似度检索相关文档片段。

3. 生成 — 以检索到的上下文为条件，生成问题的回答。

进阶的 RAG（Advanced RAG）

Naive RAG 在检索质量、响应生成质量以及增强过程中存在多个挑战。Advanced RAG 范式随后被提出，并在数据索引、检索前和检索后都进行了额外处理。通过更精细的数据清洗、设计文档结构和添加元数据等方法提升文本的一致性、准确性和检索效率。在检索前阶段则可以使用问题的重写、路由和扩充等方式对齐问题和文档块之间的语义差异。在检索后阶段则可以通过将检索

出来的文档库进行重排序避免 “Lost in the Middle ” 现象的发生。或是通过上下文筛选与压缩的方式缩短窗口长度。

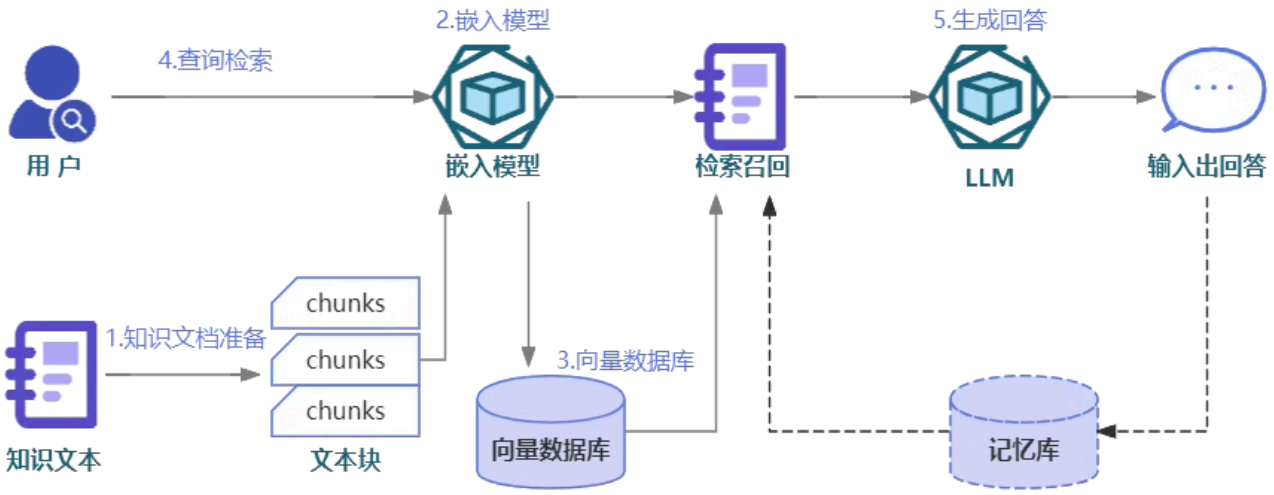
模块化 RAG（Modular RAG）

随着 RAG 技术的进一步发展和演变，新的技术突破了传统的 Naive RAG 检索 — 生成框架，基于此我们提出模块化 RAG 的概念。在结构上它更加自由的和灵活，引入了更多的具体功能模块，例如查询搜索引擎、融合多个回答。技术上将检索与微调、强化学习等技术融合。流程上也对 RAG 模块之间进行设计和编排，出现了多种的 RAG 模式。然而，模块化 RAG 并不是突然出现的，三个范式之间是继承与发展的关系。Advanced RAG 是 Modular RAG 的一种特例形式，而 Naive RAG 则是 Advanced RAG 的一种特例。

RAG框架种类	优点	缺点
初级RAG (Naive RAG)	1.采用传统的索引、检索和生成流程	1.检索质量低，可能导致 “空中
	2.直接基于用户输入进行查询	2.响应生成质量存在幻觉和询外题
	3.结合相关文档与问题形成新的提示，供大型语言模 型生成答案	3.增强过程中的集成挑战如冗余
进阶的 RAG (Advanced RAG)	1.优化数据索引，提高索引内容质量	1.高级优化增加了复杂性
	2.实施预检索和后检索方法，提高检索和生成的质量	2.对计算资源和处理能力有更高
	3.引入混合检索和索引结构优化，如图结构信息	3.需要更多的定制和调优以提高
模块化RAG (Modular RAG)	1.增加功能模块，提供多样性和灵活性	1.构建和维护模块化系统可能较
	2.适应性强，可针对特定问题上下文替换或重组模块	2. 需要仔细管理以确保模块间的
	3.采用串行或端到端训练方法，允许跨多个模块的定制	

RAG的五个基本流程

如下图所示RAG可分为5个基本流程：知识文档的准备；嵌入模型嵌入模型（embedding model）；向量数据库；查询检索和生产回答。下面会对每个环节进行详细描述：



1. 知识文档的准备

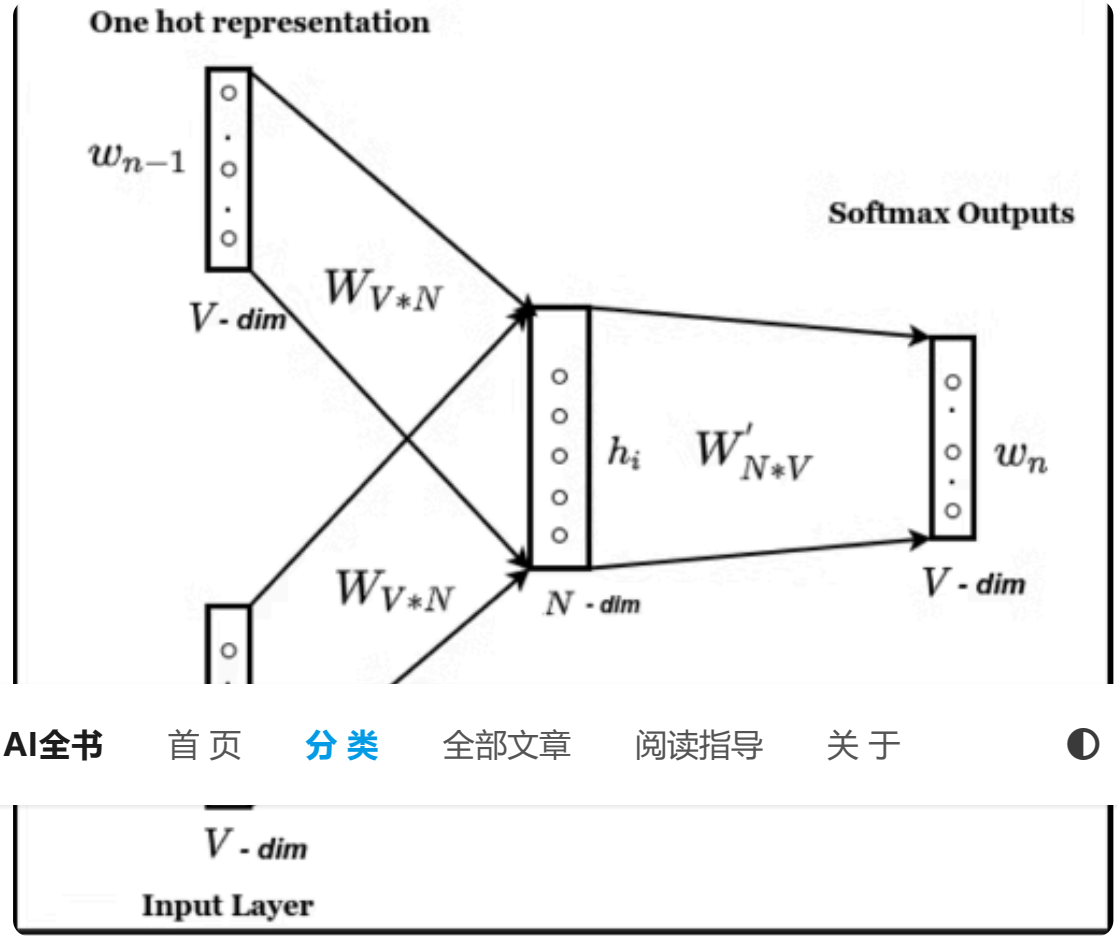
在构建一个高效的RAG系统时，首要步骤是准备知识文档。现实场景中，我们面对的知识源可能包括多种格式，如Word文档、TXT文件、CSV数据表、Excel表格，甚至是PDF文件、图片和视频等。因此，第一步需要使用专门的文档加载器（例如PDF提取器）或多模态模型（如OCR技术），将这些丰富的知识源转换为大语言模型可理解的纯文本数据。例如，处理PDF文件时，可以利用PDF提取器抽取文本内容；对于图片和视频，OCR技术能够识别并转换其中的文字信息。此外，鉴于文档可能存在过长的问题，我们还需执行一项关键步骤：文档切片。我们需要将长篇文档分割成多个文本块，以便更高效地处理和检索信息。这不仅有助于减轻模型的负担，还能提高信息检索的准确性。我们将在后文中详细讨论文档切片和其背后的逻辑。

2. 嵌入模型

嵌入模型的核心任务是将文本转换为向量形式，我们使用的日常语言中充满歧义和对表达词意无用的助词，而向量表示则更加密集、精确，能够捕捉到句子的上下文关系和核心含义。这种转换使得我们能够通过简单计算向量之间的差异来识别语义上相似的句子。

举例举例来说，如果我们想比较“苹果是一种水果”和“香蕉是黄色的”，嵌入模型可以将这些句子转换为向量，然后通过计算它们之间的相似度便可以确定它们的关联程度。那么这样的嵌入

模型是如何得到的呢，作为一个经典的例子，我们以Google开发的Word2Vec（一文看懂什么是什么是 Word2vec）模型为基础，来探讨其训练过程。



Word2Vec有两种训练方法，为简单起见我们就以上图展示的CBOW模型方法讲解。CBOW模型的核心思想是在给定一句话中心词的上下文（即该词周围的几个词）的情况下，让模型预测这个中心词。例如，假设我们有一个句子： “The cat sat on the mat” 。使用CBOW模型，我们的目标是预测中心词 “sat” 。具体的训练流程如下：

1. 对于上下文词"the", "cat", "on", "the","mat"首先生成它们的one-hot向量。One-hot向量是一个高维稀疏向量，代表词汇表中的每个单词，如果有这个词便是1，其余全是0，比如 “the” 的编码就可以是[1，0，0，0，0]

2. 接下来，将每个one-hot向量与一个大小为V×N的权重矩阵W相乘（这里V是词汇表的大小，N是词向量的维度），得到每个单词的词向量。然后，将所有这些词向量加起来并求平均，得到代表整个句子上下文的向量e（即图中的projection layer中的1×N向量）。

3. 接着，将向量e乘以另一个输出权重矩阵W'，通过SoftMax函数处理，得到一个大小为1×V的输出向量。这个向量代表了模型预测每个词是中心词的概率。

4. 最后，将模型的预测结果与实际的中心词进行比较，通过这个差值不断更新两个权重矩阵W和W'。经过足够的训练，步骤2中得到的向量e就能有效地代表整个句子的含义。

当然除了Word2Vec，还有其他高级的嵌入模型如BERT和GPT系列，它们通过更复杂的网络结构捕捉更深层次的语义关系。总之嵌入模型是连接用户查询和知识库的桥梁，确保了系统回答的准确性和相关性。

3. 向量数据库

顾名思义，向量数据库是专门设计用于存储和检索向量数据的数据库系统。在RAG系统中，通过嵌入模型生成的所有向量都会被存储在这样的数据库中。这种数据库优化了处理和存储大规模向量数据的效率，使得在面对海量知识向量时，我们能够迅速检索出与用户查询最相关的信息。

4. 查询检索

再经过上述几个步骤的准备后，我们就可以开始进行处理用户查询了。首先，用户的问题会被输入到嵌入模型中进行向量化处理。然后，系统会在向量数据库中搜索与该问题向量语义上相似的知识文本或历史对话记录并返回。

5. 生成回答

最终将用户提问和上一步中检索到的信息结合，构建出一个提示模版，输入到大语言模型中，静待模型输出答案即可。

RAG的12个优化策略

上面章节讲述的流程仅仅为基本的RAG流程，但是其中的各个环节均有着极大的优化空间。下面我们将从上述5个环节中穿插12个具体优化策略来依次讲解。下面介绍的方法均在AI开发框架langchain和LLamaIndex中有具体实现，具体操作方法可参考官方文档。

1. 数据清洗

高性能RAG系统依赖于准确且清洁的原始知识数据。一方面为了保证数据的准确性，我们需要优化文档读取器和多模态模型。特别是处理如CSV表格等文件时，单纯的文本转换可能会丢失表格原有的结构。因此，我们需引入额外的机制以在文本中恢复表格结构，比如使用分号或其他符号来区分数据。另一方面我们也需要对知识文档做一些基本数据清洗其中可以包括：

- **基本文本清理**：规范文本格式，去除特殊字符和不相关信息。除重复文档或冗余信息。

- **实体解析**：消除实体和术语的歧义以实现一致的引用。例如，将“LLM”、“大语言模型”和“大模型”标准化为通用术语。
- **文档划分**：合理地划分不同主题的文档，不同主题是集中在一处还是分散在多处？如果作为人类都不能轻松地判断出需要查阅哪个文档才能来回答常见的提问，那么检索系统也无法做到。
- **数据增强**：使用同义词、释义甚至其他语言的翻译来增加语料库的多样性。
- **用户反馈循环**：基于现实世界用户的反馈不断更新数据库，标记它们的真实性。
- **时间敏感数据**：对于经常更新的主题，实施一种机制来使过时的文档失效或更新。

2.分块处理

在RAG系统中，文档需要分割成多个文本块再进行向量嵌入。在不考虑大模型输入长度限制和成本问题情况下，其目的是在保持语义上的连贯性的同时，尽可能减少嵌入内容中的噪声，从而更有效地找到与用户查询最相关的文档部分。如果分块太大，可能包含太多不相关的信息，从而降低了检索的准确性。相反，分块太小可能会丢失必要的上下文信息，导致生成的回应缺乏连贯性或深度。在RAG系统中实施合适的文档分块策略，旨在找到这种平衡，确保信息的完整性和相关性。一般来说，理想的文本块应当在没有周围上下文的情况下对人类来说仍然有意义，这样对语言模型来说也是有意义的。

分块方法的选择

固定大小的分块：这是最简单和直接的方法，我们直接设定块中的字数，并选择块之间是否重复内容。通常，我们会保持块之间的一些重叠，以确保语义上下文不会在块之间丢失。与其他形式的分块相比，固定大小分块简单易用且不需要很多计算资源。

内容分块

顾名思义，根据文档的具体内容进行分块，例如根据标点符号（如句号）分割。或者直接使用更高级的NLTK或者spaCy库提供的句子分割功能。

递归分块

在大多数情况下推荐的方法。其通过重复地应用分块规则来递归地分解文本。例如，在langchain中会先通过段落换行符（`\n\n`）进行分割。然后，检查这些块的大小。如果大小不超过一定阈值，则该块被保留。对于大小超过标准的块，使用单换行符（`\n`）再次分割。以此类推，不断根据块大小更新更小的分块规则（如空格，句号）。这种方法可以灵活地调整块的大小。例如，对

于文本中的密集信息部分，可能需要更细的分割来捕捉细节；而对于信息较少的部分，则可以使用更大的块。而它的挑战在于，需要制定精细的规则来决定何时和如何分割文本。

从小到大分块

既然小的分块和大的分块各有各的优势，一种更为直接的解决方案是把同一文档进行从大到小所有尺寸的分割，然后把不同大小的分块全部存进向量数据库，并保存每个分块的上下级关系，进行递归搜索。但可想而知，因为我们要存储大量重复的内容，这种方案的缺点就是需要更大的储存空间。

特殊结构分块

针对特定结构化内容的专门分割器。这些分割器特别设计来处理这些类型的文档，以确保正确地保留和理解其结构。langchain提供的特殊分割器包括：Markdown文件，Latex文件，以及各种主流代码语言分割器。

分块大小的选择

上述方法中无一例外最终都需要设定一个参数——块的大小，那么我们如何选择呢？首先不同的嵌入模型有其最佳输入大小。比如Openai的text-embedding-ada-002的模型在256 或 512大小的块上效果更好。其次，文档的类型和用户查询的长度及复杂性也是决定分块大小的重要因素。处理长篇文章或书籍时，较大的分块有助于保留更多的上下文和主题连贯性；而对于社交媒体帖子，较小的分块可能更适合捕捉每个帖子的精确语义。如果用户的查询通常是简短和具体的，较小的分块可能更为合适；相反，如果查询较为复杂，可能需要更大的分块。实际场景中，我们可能还是需要不断实验调整，在一些测试中，128大小的分块往往是最佳选择，在无从下手时，可以从这个大小作为起点进行测试。

3. 嵌入模型

我们提到过嵌入模型能帮助我们把文本转换成向量，显然不同的嵌入模型带来的效果也不尽相同，例如，我们之前讨论的Word2Vec模型，尽管功能强大，但存在一个重要的局限性：其生成的词向量是静态的。一旦模型训练完成，每个词的向量表示就固定不变，这在处理一词多义的情况时可能导致问题。

比如，“我买了一张光盘”，这里“光盘”指的是具体的圆形盘片，而在“光盘行动”中，“光盘”则指的是把餐盘里的食物吃光，是一种倡导节约的行为。语义完全不一样的词向量却是固定的。相比之下，引入自注意力机制的模型，如BERT，能够提供动态的词义理解。这意味着它可以根据上下文动态地调整词义，使得同一个词在不同语境下有不同的向量表示。在之前的例子中，“光盘”这个词在两个句子中会有不同的向量，从而更准确地捕捉其语义。

有些项目为了让模型对特定垂直领域的词汇有更好的理解，会嵌入模型进行微调。但在这里我们并不推荐这种方法，一方面其对训练数据的质量有较高要求，另一方面也需要较多的人力物力投入，且效果未必理想，最终得不偿失。在这种情况下，对于具体应该如何选择嵌入模型，我们推荐参考Hugging Face推出的嵌入模型排行榜MTEB。这个排行榜提供了多种模型的性能比较，能帮助我们做出更明智的选择。同时，要注意并非所有嵌入模型都支持中文，因此在选择时应查阅模型说明。

4. 元数据

当在向量数据库中存储向量数据时，某些数据库支持将向量与元数据（即非向量化的数据）一同存储。为向量添加元数据标注是一种提高检索效率的有效策略，它在处理搜索结果时发挥着重要作用。

例如，日期就是一种常见的元数据标签。它能够帮助我们根据时间顺序进行筛选。设想一下，如果我们正在开发一款允许用户查询他们电子邮件历史记录的应用程序。在这种情况下，日期最近的电子邮件可能与用户的查询更相关。然而，从嵌入的角度来看，我们无法直接判断这些邮件与用户查询的相似度。通过将每封电子邮件的日期作为元数据附加到其嵌入中，我们可以在检索过程中优先考虑最近日期的邮件，从而提高搜索结果的相关性。

此外，我们还可以添加诸如章节或小节的引用，文本的关键信息、小节标题或关键词等作为元数据。这些元数据不仅有助于改进知识检索的准确性，还能为最终用户提供更加丰富和精确的搜索体验。

5.多级索引

元数据无法充分区分不同上下文类型的情况下，我们可以考虑进一步尝试多重索引技术。多重索引技术的核心思想是将庞大的数据和信息需求按类别划分，并在不同层级中组织，以实现更有效的管理和检索。这意味着系统不仅依赖于单一索引，而是建立了多个针对不同数据类型和查询需求的索引。例如，可能有一个索引专门处理摘要类问题，另一个专门应对直接寻求具体答案的问题，还有一个专门针对需要考虑时间因素的问题。这种多重索引策略使RAG系统能够根据查询的性质和上下文，选择最合适的索引进行数据检索，从而提升检索质量和响应速度。但为了引入多重索引技术，我们还需配套加入多级路由机制。

多级路由机制确保每个查询被高效引导至最合适的索引。查询根据其特点（如复杂性、所需信息类型等）被路由至一个或多个特定索引。这不仅提升了处理效率，还优化了资源分配和使用，确保了对各类查询的精确匹配。

例如，对于查询“最新上映的科幻电影推荐”，RAG系统可能首先将其路由至专门处理当前热点话题的索引，然后利用专注于娱乐和影视内容的索引来生成相关推荐。

总的来说，多级索引和路由技术可以进一步帮助我们对大规模数据进行高效处理和精准信息提取，从而提升用户体验和系统的整体性能。

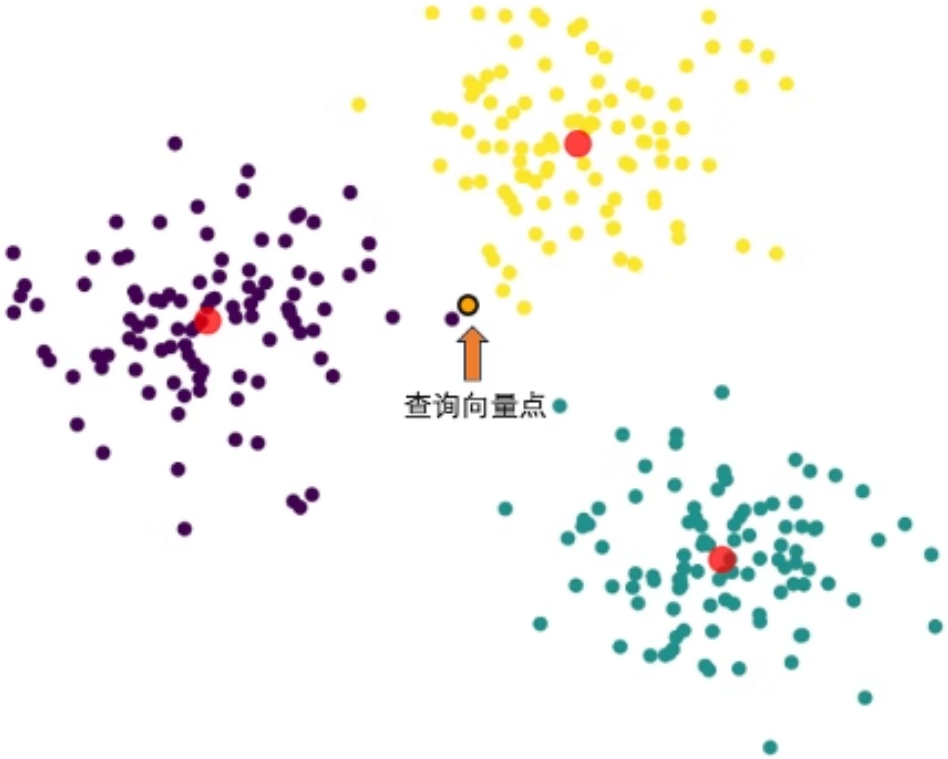
6.索引/查询算法

我们可以利用索引筛选数据，但说到底我们还是要从筛选后的数据中检索出相关的文本向量。由于向量数据量庞大且复杂，寻找绝对的最优解变得计算成本极高，有时甚至是不可行的。加之，大模型本质上并不是完全确定性的系统，这些模型在搜索时追求的是语义上的相似性——一种合理的匹配即可。从应用的角度来看，这种方法是合理的。

例如，在推荐系统中，用户不太可能察觉到或关心是否每个推荐的项目都是绝对的最佳匹配；他们更关心的是推荐是否总体上与他们的兴趣相符。因此查找与查询向量完全相同的项通常不是目标，而是要找到“足够接近”或“相似”的项，这便是最近邻搜索（Approximate Nearest Neighbor Search，ANNS）。这样做不仅能满足需求，还为检索优化提供了巨大的优化潜力。下面我们会介绍一些常见的向量搜索算法以便大家在具体使用场景中进行取舍。

聚类

当我们在网上购物时，通常不会在所有商品中盲目搜索，而是会选择进入特定的商品分类，比如“电子产品”或“服饰”，在一个更加细分的范畴内寻找心仪的商品。这个能帮我们大大缩小搜索范围。同样这种思路，聚类算法可以帮我们实现这个范围的划定。就比如说我们可以用K-mean算法把向量分为数个簇，当用户进行查询的时候，我们只需找到距离查询向量最近的簇，然后再这个簇中进行搜索。当然聚类的方法并不保证一定正确，如下图，查询距离黄色簇的中心点更近，但实际上距离查询向量最近的，即最相似的点在紫色类。

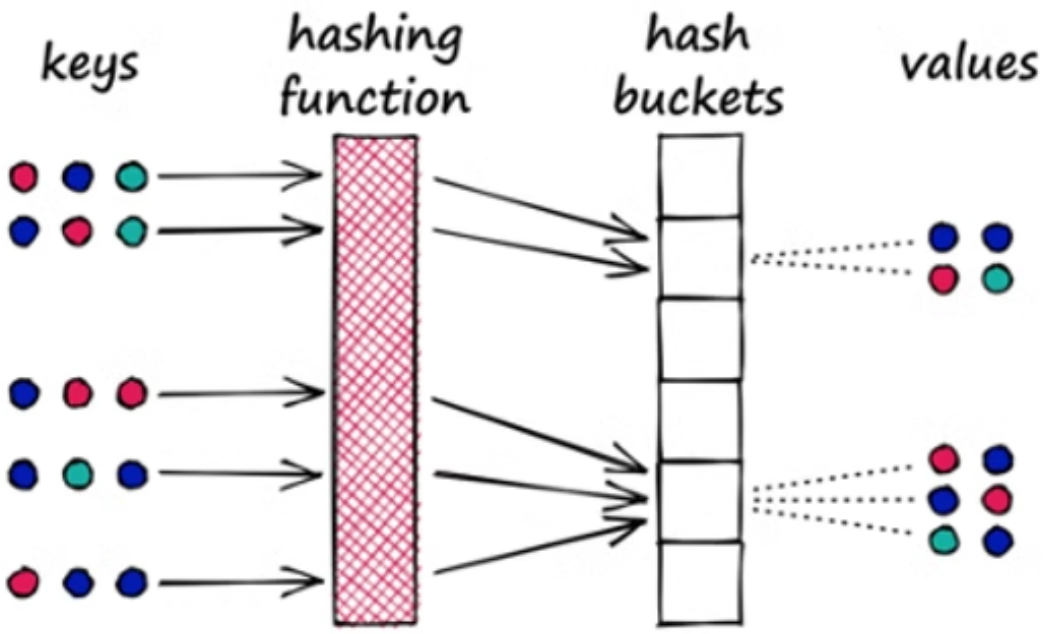


有一些缓解这个问题的方法，例如增加聚类的数量，并指定搜索多个簇。然而，任何提高结果质量的方法都不可避免地会增加搜索的时间和资源成本。实际上，质量和速度之间存在着一种权衡关系。我们需要在这两者之间找到一个最优的平衡点，或者找到一个适合特定应用场景的平衡。不同的算法也对应着不同的平衡。

可以看看《一文看懂什么是K均值聚类》。

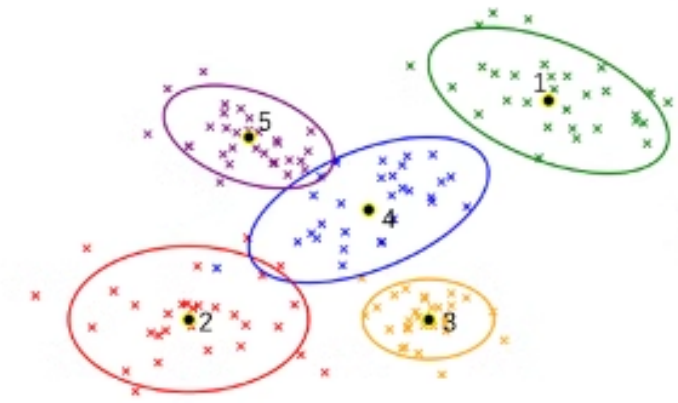
位置敏感哈希

沿着缩小搜索范围的思路，位置敏感哈希算法是另外一种实现的策略。在传统的哈希算法中，我们通常希望每个输入对应一个唯一的输出值，并努力减少输出值的重复。然而，在位置敏感哈希算法中，我们的目标恰恰相反，我们需要增加输出值碰撞的概率。这种碰撞正是分组的关键，哈希值相同的向量将被分配到同一个组中，也就是同一个“桶”里。此外，这种哈希函数还需满足另一个条件：空间上距离较近的向量更有可能被分入同一个桶。这样在进行搜索时，只需获取目标向量的哈希值，找到相应的桶，并在该桶内进行搜索即可。



量化乘积

上面我们介绍了两种牺牲搜索质量来提高搜索速度的方法，但除了搜索速度外，内存开销也是一个巨大挑战。在实际应用场景中，每个向量往往都有上千个维度，数据数量可达上亿。每条数据都对应着一个实际的的信息，因此不可能删除数据来减少内存开销，那唯一的选择只能是把每个数据本身大小缩减。有一种乘积量化的方法可以帮我们完成这点。图像有一种有损压缩的方法是把一个像素周围的几个像素合并，来减少需要储存的信息。同样我们可以在聚类的方法之上改进一下，用每个簇的中心点来代替簇中的数据点。虽然这样我们会丢失向量的具体值信息，但考虑到聚类中心点和簇中向量相关程度，再加上可以不断增加簇的数量来减少信息损失，所以很大程度上我们可以保留原始点的信息。而这样做带来的好处是十分可观的。如果我们给这些中心点编码，我们就可以用单个数字储存一个向量来减少存储的空间。而我们把每个中心向量值和他的编码值记录下来形成一个码本，这样每次使用某个向量的时候，我们只需用他的编码值通过码本找到对应的的中心向量的具体值，虽然这个向量已经不再是当初的样子了，但就像上面所说，问题不大。而这个把向量用其所在的簇中心点表示的过程就是量化。



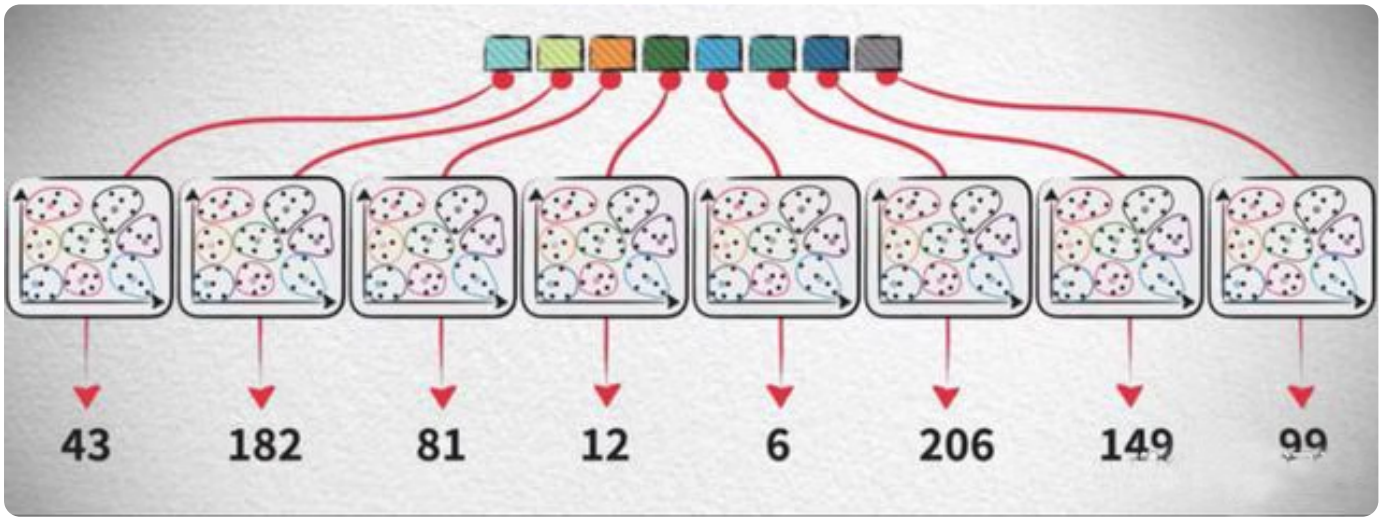
码本

编码	向量坐标
1	(2,2)
2	(8,8)
3	(5,5)
4	(3,7)
5	(6,2)

内存

向量	编码
✖	2
✖	2
✖	1
✖	4
✖	4
✖	3

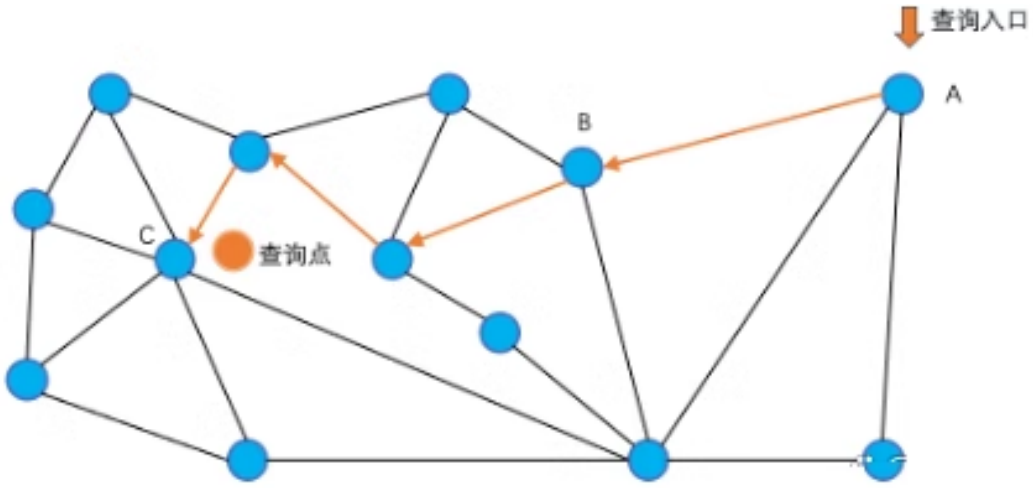
但是你可能会发现码本又引入了额外的开销。在真实的场景中因为维度爆炸的问题，更高的维度中数据分布更加稀疏，所以我们需要更多的聚类数量来减少丢失的细节，就比如一个128维的空间，我们需要2的64次方个聚类中心，所以最终这个码本所需的内存会变得非常恐怖，甚至超越量化本身所节省下来的内存。解决这个问题的办法是把高维的向量分割成多个低维的子向量，然后在这些低维的子向量中独立量化。比如把128维的向量分解成8个16维的子向量，然后独立的在8个独立的子空间进行量化形成自己子码本。



每个16维的空间只需要256个聚类就可以得到不错的效果，并且每个子码本将变得很小，即便8个子码本加在一起的大小也仍然足够的小，实际上我们是在把码本大小增长从指数模型分解为加法模式。最后，我们还可以把前面的提速度方法和减内存的乘积量化法结合使用，同时实现速度和内存的优化。

分层导航小世界

从客户的角度来看，内存开销可能并不是最重要的考量因素。他们更加关注的是应用的最终效果，也就是回答用户问题的速度和质量。导航小世界（NSW）算法正是这样一种用内存换取更快速度和更高质量的实现方式。这个算法的思路和“六度分割理论”类似——你和任何一个陌生人之间最多只隔六个人，也就是说，最多通过六个人你就能够认识任何一个陌生人。我们可以将人比作向量点，把搜索过程看作是从一个人找到另一个人的过程。在查询时，我们从一个选定的起始点A开始，然后找到与A相邻且最接近查询向量的点B，导航到B点，再次进行类似的判断，如此反复，直到找到一个点C，其所有相邻节点都没有比它更接近目标。最终这个点C便是我们要找的最相似的向量。



7. 查询转换

在RAG系统中，用户的查询问题被转化为向量，然后在向量数据库中进行匹配。不难想象，查询的措辞会直接影响搜索结果。如果搜索结果不理想，可以尝试以下几种方法对问题进行重写，以提升召回效果：

结合历史对话的重新表述

在向量空间中，对人类来说看似相同的两个问题其向量大小并不一定很相似。我们可以直接利用LLM 重新表述问题来进行尝试。此外，在进行多轮对话时，用户的提问中的某个词可能会指代上文中的部分信息，因此可以将历史信息和用户提问一并交给LLM重新表述。

假设文档嵌入（HyDE）

HyDE的核心思想是接收用户提问后，先让LLM在没有外部知识的情况下生成一个假设性的回复。然后，将这个假设性回复和原始查询一起用于向量检索。假设回复可能包含虚假信息，但蕴含着LLM认为相关的信息和文档模式，有助于在知识库中寻找类似的文档。

退后提示（Step Back Prompting）

如果果原始查询太复杂或返回的信息太广泛，我们可以选择生成一个抽象层次更高的“退后”问题，与原始问题一起用于检索，以增加返回结果的数量。例如，原问题是“桌子君在特定时期去了哪所学校”，而退后问题可能是关于他的“教育历史”。这种更高层次的问题可能更容易找到答案。

多查询检索/多路召回（Multi Query Retrieval）

使用LLM生成多个搜索查询，特别适用于一个问题可能需要依赖多个子问题的情况。

通过这些方法，RAG系统能够更精准地处理和响应复杂的用户查询，从而提升整体的搜索效率和准确性。

8. 检索参数

终于我们把查询问题准备好了，可以进入向量数据库进行检索。在具体的检索过程中，我们可以根据向量数据库的特定设置来优化一些检索参数，以下是一些常见的可设定参数：

稀疏和稠密搜索权重

稠密搜索即通过向量进行搜索。然而，在某些场景下可能存在限制，此时可以尝试使用原始字符串进行关键字匹配的稀疏搜索。一种有效的稀疏搜索算法是最佳匹配25（BM25），它基于统计输入短语中的单词频率，频繁出现的单词得分较低，**而稀有的词被视为关键词，得分会较高**。我们可以结合稀疏和稠密搜索得出最终结果。向量数据库通常允许设定两者对最终结果评分的权重比例，如0.6表示40%的得分来自稀疏搜索，60%来自稠密搜索。

结果数量（topK）

检索结果的数量是另一个关键因素。足够的检索结果可以确保系统覆盖到用户查询的各个方面。在回答多方面或复杂问题时，更多的结果提供了丰富的语境，有助于RAG系统更好地理解问题的上下文和隐含细节。但需注意，结果数量过多可能导致信息过载，降低回答准确性并增加系统的时间和资源成

相似度度量方法

计算两个向量相似度的方法也是一个可选参数。这包括使用欧式距离和Jaccard距离计算两个向量的差异，以及利用余弦相似度衡量夹角的相似性。通常，余弦相似度更受青睐，因为它不受向量长度的影响，只反映方向上的相似度。这使得模型能够忽略文本长度差异，专注于内容的语义相似性。需要注意的是，并非所有嵌入模型都支持所有度量方法，具体可参考所用嵌入模型的说明。

9. 高级检索策略

终于我们来到最为关键和复杂的步骤——在向量数据库检索之上如何具体开发或改进整个系统的策略。这部分的内容足够写成一篇独立文章。为了保持简洁，我们只讨论一些常用或者新提出的策略。

上下文压缩

我们提到过当文档文块过大时，可能包含太多不相关的信息，传递这样的整个文档可能导致更昂贵的LLM调用和更差的响应。上下文压缩的思想就是通过LLM的帮助根据上下文对单个文档内容进行压缩，或者对返回结果进行一定程度的过滤仅返回相关信息。

句子窗口搜索

相反，文档文块太小会导致上下文的缺失。其中一种解决方案就是窗口搜索，该方法的核心思想是当提问匹配好分块后，将该分块周围的块作为上下文一并交给LLM进行输出，来增加LLM对文档上下文的理解。

父文档搜索

无独有偶，父文档搜索也是一种很相似的解决方案，父文档搜索先将文档分为尺寸更大的主文档，再把主文档分割为更短的子文档两个层级，用户问题会与子文档匹配，然后将该子文档所属的主文档和用户提问发送给LLM。

自动合并

自动合并是在父文档搜索上更进一步的复杂解决方案。同样地，我们先对文档进行结构切割，比如将文档按三层树状结构进行切割，顶层节点的块大小为1024，中间层的块大小为512，底层的叶子节点的块大小为128。而在检索时只拿叶子节点和问题进行匹配，当某个父节点下的多数叶子节点都与问题匹配上则将该父节点作为结果返回。

多向量检索

多向量检索同样会给一个知识文档转化成多个向量存入数据库，不同的是，这些向量不仅包括文档在不同大小下的分块，还可以包括该文档的摘要，用户可能提出的问题等等有助于检索的信息。在使用多向量查询的情况下，每个向量可能代表了文档的不同方面，使得系统能够更全面地考虑文档内容，并在回答复杂或多方面的查询时提供更精确的结果。例如，如果查询与文档的某个具体部分或摘要更相关，那么相应的向量就可以帮助提高这部分内容的检索排名。

多代理检索

多代理检索，简而言之就是选取我们提及的12大优化策略中的部分交给一个智能代理合并使用。就比如使用子问题查询，多级索引和多向量查询结合，先让子问题查询代理把用户提问拆解为多个小问题，再让文档代理对每个字问题进行多向量或多索引检索，最后排名代理将所有检索的文

档总结再交给LLM。这样做的好处是可以取长补短，比如，子问题查询引擎在探索每个子查询时可能会缺乏深度，尤其是在相互关联或关系数据中。相反，文档代理递归检索在深入研究特定文档和检索详细答案方面表现出色，以此来综合多种方法解决问题。需要注意的是现在网络上存在不同结构的多代理检索，具体在多代理选取哪些优化步骤尚未有确切定论，我们可以结合使用场景进行探索。

Self-RAG

自反思搜索增强是一个全新RAG框架，其与传统RAG最大的区别在于通过检索评分（令牌）和反思评分（令牌）来提高质量。它主要分为三个步骤：检索、生成和批评。Self-RAG首先用检索评分来评估用户提问是否需要检索，如果需要检索，LLM将调用外部检索模块查找相关文档。接着，LLM分别为每个检索到的知识块生成答案，然后为每个答案生成反思评分来评估检索到的文档是否相关，最后将评分高的文档当作最终结果一并交给LLM。

10.重排模型（Re-ranking）

在完成语义搜索的优化步骤后，我们能够检索到语义上最相似的文档，但不知你是否注意到一个关键问题：语义最相似是否总代表最相关？答案是不一定。例如，当用户查询“最新上映的科幻电影推荐”时，可能得到的结果是“科幻电影的历史演变”，虽然从语义上这与科幻电影相关，但并未直接回应用户关于最新电影的查询。

重排模型可以帮助我们缓解这个问题，重排模型通过对初始检索结果进行更深入的相关性评估和排序，确保最终展示给用户的结果更加符合其查询意图。这一过程通常由深度学习模型实现，如Cohere模型。这些模型会考虑更多的特征，如查询意图、词汇的多重语义、用户的历史行为和上下文信息等。

举个例子，对于查询“最新上映的科幻电影推荐”，在首次检索阶段，系统可能基于关键词返回包括科幻电影的历史文章、科幻小说介绍、最新电影的新闻等结果。然后，在重排阶段，模型会对这些结果进行深入分析，并将最相关、最符合用户查询意图的结果（如最新上映的科幻电影列表、评论或推荐）排在前面，同时将那些关于科幻电影历史或不太相关的内容排在后面。这样，重排模型就能有效提升检索结果的相关性和准确性，更好地满足用户的需求。

在实践中，使用RAG构建系统时都应考虑尝试重排方法，以评估其是否能够提高系统性能。

11. 提示词

大型语言模型的解码器部分通常基于给定输入来预测下一个词。这意味着设计提示词或问题的方式将直接影响模型预测下一个词的概率。这也给了我们一些启示：通过改变提示词的形式，可以

有效地影响模型对不同类型问题的接受程度和回答方式，比如修改提示语，让LLM知道它在做什么工作，是十分有帮助的。

为了减少模型产生主观回答和幻觉的概率，一般情况下，**RAG系统中的提示词中应明确指出回答仅基于搜索结果**，不要添加任何其他信息例如，可以设置提示词如：

> “你是一名智能客服。你的目标是提供准确的信息，并尽可能帮助提问者解决问题。你应保持友善，但不要过于啰嗦。请根据提供的上下文信息，在不考虑已有知识的情况下，回答相关查询。”

当然你也可以根据场景需要，也可以适当让模型的回答融入一些主观性或其对知识的理解。此外，使用少量样本（few-shot）的方法，将想要的问答例子加入提示词中，指导LLM如何利用检索到的知识，也是提升LLM生成内容质量的有效方法。这种方法不仅使模型的回答更加精准，也提高了其在特定情境下的实用性。

12. 大语言模型

终于我们来到最后一步——LLM生成回答。LLM是生成响应的核心组件。与嵌入模型类似，可以根据自己的需求选择LLM，例如开放模型与专有模型、推理成本、上下文长度等。此外，可以使用一些LLM开发框架来搭建RAG系统，比如，LlamaIndex或LangChain。这两个框架都拥有比较好用的debugging工具，可以让我们定义回调函数，查看使用了哪些上下文，检查检索结果来自哪个文档等等。

总结

本文介绍了什么是RAG，RAG的特点以及发展阶段，然后讲了RAG的流程包括12个优化策略，前端部分适合大众阅读，后面部分专业性比较强，适合技术实践者阅读。

人工智能技术关键词

课程 工具使用 理论 自媒体 产品经理 理论研究

Further Reading

Apr 27, 2024

学习AI必须了解的技术性关键词一览

学习AI必须了解的技术性关键词一览，关键词科普，即使不从事AI也得了解下这些词。

Apr 15, 2024

一文看懂什么是智能体（AI Agent）

本文介绍智能体的起源、定义、特征、组成单元，适用的工作，最后面像开发者介绍一些Agent实践开源框架。

Apr 6, 2024

一文看懂什么是RAG（检索增强生成）

一文看懂什么是RAG，RAG和大模型什么关系，RAG的发展阶段，五个基本流程和12个优化策略

OLDER
一文看懂什么是通用人工智能（AGI）

NEWER
19类Agent（智能体）框架对比