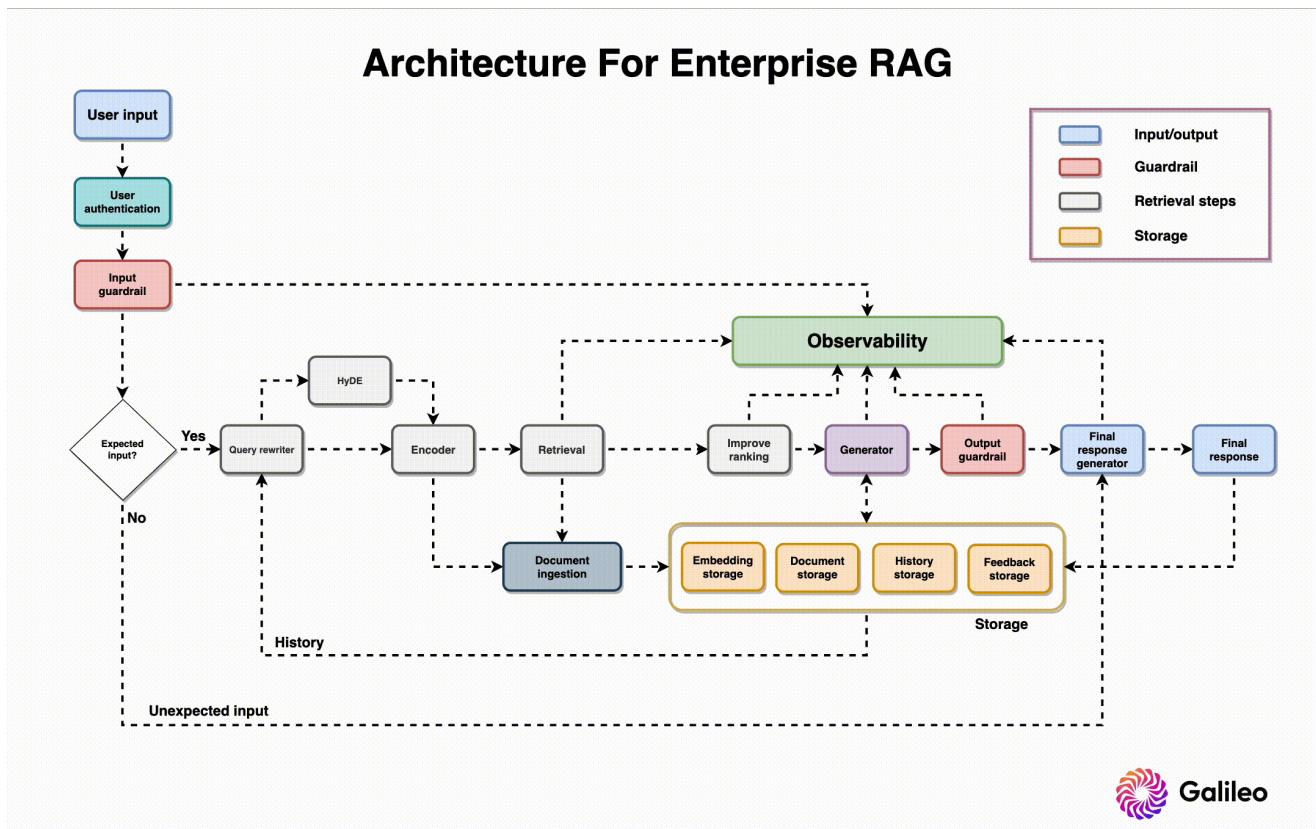


原文地址: [Mastering RAG: How To Architect An Enterprise RAG System](#)

欢迎继续关注我们的精通 RAG 系列课程！我们一起努力一把，深入企业级 RAG 系统构建的纷繁世界。

当我们在浩如烟海的互联网上搜索与 RAG 系统相关的信息时，琳琅满目的文章多是介绍基本构建方法，而如何打造一个坚不可摧的企业级解决方案则往往像是一块未解之谜。很多构建者在构建 RAG 系统的过程中，甚至连最关键的决定是什么都没弄清楚...

不过，本系列博文不仅仅停留在理论上，更是一本指导实践的工具书！我们不仅讨论如何的设置安全保护来确保系统安全，还会探讨查询重写如何提升用户体验，旨在为你提供实用的建议和现实案例解析。无论你是资深开发人员还是技术团队的领头者，我们一起深入挖掘，探索先进企业级 RAG 系统构建的复杂内涵。



在我们深入探讨 RAG 架构之前，让我先介绍一个最新[研究](#)，这项研究调查了构建 RAG 系统时的常见故障点。研究人员分析了三个跨领域的案例研究，找出了七个共有的故障点。

构建 RAG 系统的挑战

案例研究

Case Study	Domain	Doc Types	Dataset Size	RAG Stages	Sample Questions
Cognitive Reviewer*	Research	PDFs	(Any size)	Chunker, Rewriter, Retriever, Reader	What are the key points covered in this paper?
AI Tutor*	Education	Videos, HTML, PDF	38	Chunker, Rewriter, Retriever, Reader	What were the topics covered in week 6?
BioASQ	Biomedical	Scientific PDFs	4017	Chunker, Retriever, Reader	Define pseudotumor cerebri. How is it treated?

Table 1: A summary of the RAG case studies presented in this paper. Case studies marked with a * are running systems currently in use.

认知审校系统

认知审校系统是设计来帮助研究者分析科学文件的 RAG 系统。研究者定义研究问题或目标，并上传相关研究文献集合。接着，系统会根据所陈述目标对所有文档进行排名，以便研究者手动审核。研究者也可以直接向整个文档集提问。

AI 辅导系统

AI 辅导系统，让学生能够就学习单元提出问题并得到答案，答案来自学习内容资源。学生还能通过查阅资源列表来验证答案。该系统融合到 Deakin 大学的教学管理系统中，对所有学习材料进行索引，包含 PDF 文件、视频和文字文档。利用 Whisper 深度学习模型，系统在文本块切分前对视频内容进行文字转录。RAG 流程中融入了用于查询概化的重写器，聊天界面还利用先前对话作为每个问题的背景上下文。

生物医学问答

在生物医学问答的案例研究中，研究者使用 BioASQ 数据集建立了一个 RAG 系统，该数据集包含问题、文档链接和答案。这份由生物医学专家准备的数据集包括了特定领域的问答对。问题的答案可以是是/否，也可以是文本摘要、事实点或列表形式的。

RAG 系统的 7 大故障点

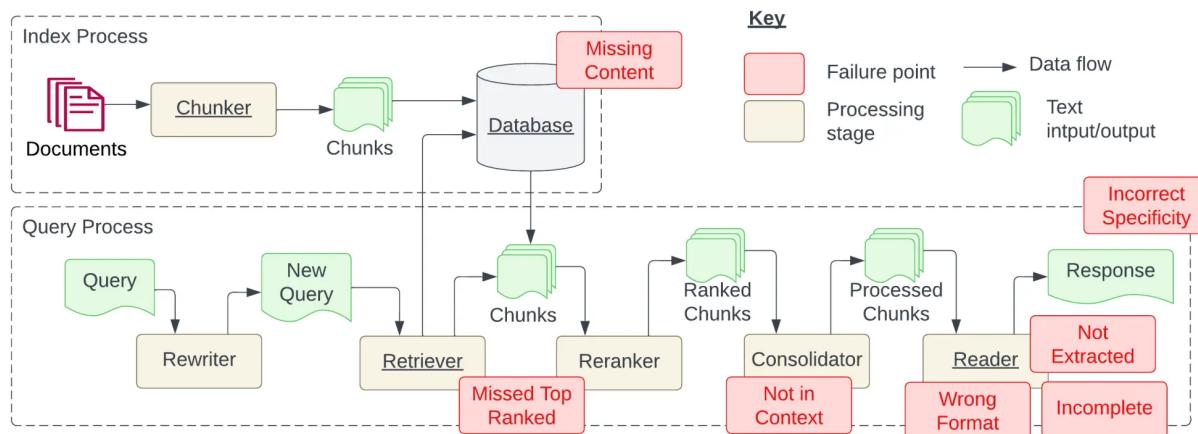


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

通过这些案例研究，研究者找出了在设计 RAG 系统过程中常遇到的七个问题点。

缺失内容（故障点1）

当提出的问题无法根据现有文档回答时，理想情况下，RAG 系统会回应：“很抱歉，我不知道。”然而，如果问题涉及内容没有明确答案，系统可能误提供答复。

错过排名靠前的文档（故障点2）

尽管答案在文档中，系统未能将其排名至足够高的位置以提供给用户。实际操作中，尽管理论上所有文档都会被进行排名并应用于后续步骤，但实际上只有排名前 K 的文档会返回给用户，K 是根据性能选定的数值。

脱离上下文 - 合并策略的局限性（故障点3）

数据库检索到包含答案的文档，但答案没能成为生成回复的上下文。这通常发生在返回了大量文档，导致在合并过程中相关答案的检索受阻。

未提取答案（故障点4）

答案虽然在上下文中，但模型未能提取正确的信息，这类问题通常由于上下文信息过多干扰或信息矛盾导致。

格式错误（故障点5）

当问题要求提取特定格式的信息，比如表格或列表时，模型并未遵循这样的指令。

不精确（故障点6）

答案虽然包含在回复中，但缺少所需的具体性，或过于具体，未能满足用户的需求。这种情况出现在 RAG 系统设计者对给定问题有预设的答案时，例如教师寻找教学内容时。在这种情况下，应当提供特定的教学内容和答案。不精确也发生在用户不清楚如何准确提问而提出过于笼统的问题时。

不完整（故障点7）

答案虽然准确，却缺少一些信息，即使这些信息在上下文中已经存在且可提取。例如，“文档 A、B 和 C 涵盖了哪些关键点？”如果将问题分开询问可能会得到更好的解答。

FP	Lesson	Description	Case Studies
FP4	Larger context get better results (Context refers to a particular setting or situation in which the content occurs)	A larger context enabled more accurate responses (8K vs 4K). Contrary to prior work with GPT-3.5 [13]	AI Tutor
FP1	Semantic caching drives cost and latency down	RAG systems struggle with concurrent users due to rate limits and the cost of LLMs. Prepopulate the semantic cache with frequently asked questions [1]. Research suggests fine-tuning LLMs reverses safety training [11], test all fine-tuned LLMs for RAG system.	AI Tutor
FP5-7	Jailbreaks bypass the RAG system and hit the safety training.		AI Tutor
FP2, FP4	Adding meta-data improves retrieval.	Adding the file name and chunk number into the retrieved context helped the reader extract the required information. Useful for chat dialogue.	AI Tutor
FP2, FP4-7	Open source embedding models perform better for small text.	Open source sentence embedding models performed as well as closed source alternatives on small text.	BioASQ, AI Tutor
FP2-7	RAG systems require continuous calibration.	RAG systems receive unknown input at runtime requiring constant monitoring.	AI Tutor, BioASQ
FP1, FP2	Implement a RAG pipeline for configuration.	A RAG system requires calibrating chunk size, embedding strategy, chunking strategy, retrieval strategy, consolidation strategy, context size, and prompts.	Cognitive Reviewer, AI Tutor, BioASQ
FP2, FP4	RAG pipelines created by assembling bespoke solutions are suboptimal.	End-to-end training enhances domain adaptation in RAG systems [18].	BioASQ, AI Tutor
FP2-7	Testing performance characteristics are only possible at runtime.	Offline evaluation techniques such as G-Evals [14] look promising but are premised on having access to labelled question and answer pairs.	Cognitive Reviewer, AI Tutor

Table 2: The lessons learned from the three case studies with key takeaways for future RAG implementations

下方表格罗列的是他们通过解决上述问题中学到的经验教训。我们在构建我们的企业级 RAG 系统时会铭记这些宝贵的教训。

打造企业级 RAG 系统的实用指南

在确认了设计 RAG 系统时所遇到的普遍问题后，我们现在来详细讨论每个组件的设计需求和角色，以及它们的构建最佳实践。上文展示的 RAG 系统架构图提供了每个组件使用的背景和方式。

用户认证

这是系统的出发点 — 我们的首个组件！在用户开始与聊天机器人交互前，我们需要出于多种原因对其进行认证。对企业系统而言，确保安全和个性化服务是必须的条件。

访问控制

用户认证确保只有授权用户获取系统访问权。它能控制谁能与系统交互以及他们允许执行的操作。

数据安全

保护敏感数据是至关重要的。用户认证可防止未授权人士获取机密信息，避免数据泄露和未授权操作。

用户隐私

认证功能保障用户隐私，确保只有真正的用户能获取他们的个人信息和账户详情。这对于树立用户信任非常关键。

法律合规

许多法制区域和行业均有规定，要求公司必须实施适当的用户认证措施保护数据和用户隐私。遵循这些规定可避免法律问题和潜在的罚款。

责任追踪

用户认证通过将系统内的行动与特定用户账户连接起来，确保责任可追踪。这对于审计和追踪用户行为至关重要，帮助识别和应对任何安全事件或可疑行动。

个性化和定制化

认证允许系统识别独立用户，使得用户体验能够个性化和定制。这可能包括定制内容、偏好设置等。

像 [AWS Cognito](#) 或 [Firebase Authentication](#) 这样的服务有助于用户在移动和网络应用程序中轻松添加注册和认证功能。

输入安全保护

我们必须阻止可能造成伤害的用户输入，或包含个人私密信息。最近的研究显示，对 LLM 的 [系统越权](#) 非常容易发生。这时，输入安全保护就显得特别重要。让我们来看看可能需要采取保护措施的不同场景。

匿名处理

输入保护条款可以对个人身份信息（PII）进行匿名处理或删减，如姓名、地址或联系方式。这有助于保护隐私，并防止试图泄露敏感信息的恶意行为。

禁止特定字串

禁止可能导致 SQL 注入、跨站脚本（XSS）或其他注入攻击的特定字串或模式，可防止系统安全弱点或不当行为的发生。

限制话题

为了过滤与某些不适宜、冒犯或违反社区准则相关的特定话题，我们需要筛选掉涉及仇恨言论、歧视或露骨内容的讨论和输入。

限制代码

重要的是要防止可执行代码的注入，这可能危及系统的安全并导致代码注入攻击。

限制语言

确认输入文本是正确的语言或脚本，以防止潜在的误解或处理错误。

检测提示注入

减轻误导性或有害提示注入的尝试，这些注入可能会操纵系统或以意料之外的方式影响 LLM 的行为。

限制令牌数

对用户输入实施最大令牌或字符上限有助于避免资源消耗过大，防止服务拒绝（DoS）攻击。

检测有害内容

实施有害内容筛选器，识别并阻止包含有害或辱骂性语言的输入。

为了保护您的 RAG 系统不受这些潜在威胁，您可以利用 Meta 的 [Llama Guard](#)，您可以选择自己托管或使用像 [Sagemaker](#) 这样的托管服务。但是，别指望它能完美检测出有害内容。

查询改写器

一旦查询通过了输入安全检查，我们便将其发往查询改写器。用户的查询有时可能含糊不清或需要更多上下文才能更好地理解用户意图。查询改写是一个解决此问题的技巧。它涉及改造用户查询，以提升清晰度、精度和相关性。我们来看看一些最流行的改写技术。

基于历史重写

在这个方法中，系统利用用户的历史查询来理解对话上下文，并使后续的查询更加明确。我们以信用卡查询为例。

查询历史：

"你有几张信用卡？"
"白金和金信用卡有年费吗？"
"比较两者的功能。"

我们需要通过用户的查询历史识别上下文演变，确切把握用户的意图以及查询间的关系，并生成符合不断发展的上下文的查询。

改写后的查询："比较白金信用卡和金信用卡的功能。"

生成子查询

复杂的查询可能因检索问题而难以回答。简化这个任务的方法是将查询分解成更具体的子查询。这有助于检索用于生成答案所需的正确上下文。LlamaIndex 称之为 [子问题查询引擎](#)。

面对像 "比较白金和金信用卡的功能" 这样的查询时，系统会针对原始查询中提到的每种卡片生成子查询。

改写后的子查询：

1. "白金信用卡有哪些功能？"
2. "金信用卡有哪些功能？"

创建相似查询

为了提升检索正确文献的可能性，我们会基于用户的输入生成相似的查询，这样做能够弥补语义或词汇匹配在检索过程中的限制。

举个例子，当用户询问信用卡特性时，系统会生成相关的查询。这就涉及到运用同义词、相关术语或特定领域的知识，来创造与用户意图相一致的查询。

例如，用户问：“我想了解白金信用卡”，系统则生成：“告诉我关于白金信用卡的好处。”

编码器

确定了原始查询和改写后的查询，我们会将它们编码成向量（即数字列表）进行检索。**选择一个合适的编码器或许是构建 RAG 系统中最关键的决策。**以下是选择文本编码器时需要考虑的因素和原因。

利用 MTEB 基准测试

为了综合评估编码器的能力，[Massive Text Embedding Benchmark](#) (MTEB) 是一个很好的资源。它允许我们基于向量维度、平均检索性能以及模型尺寸来挑选编码器。尽管 MTEB 提供有价值的见解，但由于没有普适的评价标准，并且模型训练数据的细节可能并不透明，我们在解读这些结果时应持谨慎态度。

MTEB 不仅透露了像 OpenAI、Cohere 和 Voyager 这样的流行嵌入技术的性能，还显示出一些开源模型的性能与它们相差无几。然而，这些结果只提供了一个大概概述，并不能精确反映这些嵌入技术在特定领域中的实际表现。因此，在最终选择前，针对自己的数据集进行彻底评估非常必要，有强调了定制评估方法的重要。

自定义评估

编码器可能无法始终提供最优性能，尤其是在处理敏感信息时。因此，在这种情况下，自定义评估方法就显得非常关键。以下是几种进行自定义评估的方法：

通过标注评估

生成专用的数据集并进行标注以获得黄金标准。标注完成后，利用检索度量指标比如平均倒数排名 (MRR) 和归一化折扣累积增益 (NDCG)，量化地评估不同编码器的性能。

通过模型评估

类似于标注方法，紧接着一个数据生成过程，不过是使用大语言模型或交叉编码器进行评估。这样可以在所有编码器中设立一个相对的排名。在这之后对前三的编码器进行人工评估，以获得精确的性能指标。

通过聚类评估

采用不同聚类技术，分析在各个程度的 Silhouette 分数下数据聚类的覆盖量，以此指示聚类内向量的相似度。试验不同的算法诸如 HDBSCAN，并调整它们的参数以选出性能最佳的编

码器。基于聚类的评估为数据点的分布和分组提供了有价值的洞察，有助于选择与特定度量对齐的编码器。

选择文本编码器时要考虑的

在选择编码器时，你需要在私有编码器和公共编码器之间做出选择。也许你会倾向于使用私有编码器，因为它们使用起来很方便，但是两者之间有一些特定的权衡需要考虑。这是一个影响系统性能和响应延迟的重要决策。

查询成本

为了确保语义搜索的用户体验流畅，需要依赖嵌入式 API 服务的高可用性。OpenAI 和类似的服务提供者提供可靠的 API，省去了托管管理的需要。然而，选择一个开源模型，则需要根据模型大小和延迟需求进行工程上的努力。小型模型（最多110M参数）可以通过CPU实例来托管，而大型模型则可能需要GPU服务以满足延迟需求。

索引成本

打造语义搜索功能涉及到文档索引，这会产生一笔不小的成本。由于索引和查询使用同一个编码器，故索引成本与选定的编码器服务紧密相关。为了方便服务重置或重新索引到另一个向量数据库，建议将嵌入式向量分开存储。如果忽略这一点，将需要重新计算已有的嵌入式向量。

储存成本

对于索引了数百万向量的应用程序，向量数据库的存储成本是一个重要的考量点。存储成本随向量维度线性增加，而OpenAI的1526维向量带来了最高的存储成本。为了估算储存成本，需要计算文档的平均单位（词组或句子）数量，并据此外推。

语言支持

为了支持非英语语言，你可以选择使用多语言编码器或者配合英语编码器的翻译系统。

搜索延迟

语义搜索的延迟随着嵌入式向量的维度线性增长。为了降低延迟，更倾向于选择较低维度的向量。

隐私

在金融和医疗等敏感领域，严格的数据隐私要求可能会限制使用OpenAI之类的服务。

文档摄入

文档摄入系统负责管理数据的处理和持久化。在索引过程中，每个文档被拆分成更小的片段，然后通过嵌入式模型转换成嵌入式向量。最后将原始片段和嵌入式向量一起在数据库中索引。以下为文档摄入系统的组成部分。

文档解析器

文档解析器在从各种格式的文档中提取结构化信息方面起着核心作用，特别是在处理不同格式上。这包括诸如解析可能含有图片和表格的 PDF 文档等任务。

文档格式

文档解析器必须擅长处理各种文档格式，例如 PDF、Word、Excel 等，确保能够灵活适应不同的文档处理需求。这涉及到识别并处理文档中嵌入的内容，如超链接、多媒体元素或注释，从而提供文档内容的全面展现。

表格识别

对文档中的表格进行识别和数据提取至关重要，尤其是在报告或研究论文中，这有助于保持信息结构的完整性。比如，提取与表格相关的元数据（如标题和行列信息）可以加深对文档组织框架的理解。像 [Table Transformer](#) 这样的模型对于这项任务可能很有帮助。

图像识别

对文档内的图像应用光学字符识别（OCR）技术是为了识别和提取文本，使得其可以被索引和后续检索。

元数据提取

元数据指的是除主要内容以外的文档相关信息。它包括作者、创建日期、文档类型、关键词等详情。元数据为文档提供了宝贵的上下文信息，并通过考虑元数据属性帮助组织文档，提升搜索结果的相关性。元数据可以通过自然语言处理/光学字符识别流程提取，并作为特殊字段与文档一同被索引。

文本块分割器

你决定如何分割长文本（标记化）会影响嵌入式向量的质量和搜索系统的性能。如果文本块过小，则某些问题无法得到回答；如果文本块过大，则答案会包含不必要的噪声。利用[摘要](#)等技术可以帮助减少噪声、文本大小、编码成本和储存成本。文本块分割是一个重要但常被忽略的话题。它可能需要专业知识，类似于特征工程。例如，在对 Python 代码库进行分块时，可能会使用诸如 def/class 这样的前缀。

点击这个视频链接可以更深入地了解文本块分割。

索引器

索引器的任务你可能已经猜到了——它负责建立文档索引，这是一个结构化的数据结构（尝试快速说三遍……）。索引器促进了高效的搜索和检索操作。高效索引非常关键，因为它直接影响文档检索的速度和准确性。它的工作包括将文本块或标记映射到文档集合中对应的位置。索引器在文件检索过程中承担重要任务，包括创建索引和添加、更新或删除文档。

作为 RAG 系统的关键组件，索引器面临各种挑战，这些挑战可能影响系统的整体效率和性能。

可伸缩性问题

随着文档数量的增加，保持高效和快速的索引能力变得越来越具有挑战性。当系统难以处理越来越多的文档时，就可能出现可扩展性问题，导致索引和检索速度变慢。

实时索引更新

实时保持索引的更新可能会遇到挑战，尤其是在文档经常增加、更新或删除的系统中。确保实时 API 和索引机制能够无缝运行且不降低系统性能是一个持续存在的挑战。

一致性和原子性

在并发的文档更新或改动面前保持一致性和原子性可能相当复杂。即使在同时发生变化的情况下，确保索引更新能保持数据的完整性，这需要精心设计和执行。

优化存储空间

索引大量文档可能导致存储需求巨大。在确保索引保持可访问和响应性的同时，优化存储空间是一个持续的挑战，尤其是在存储成本受关注的情况下。

安全性和访问控制

实施合适的安全措施和访问控制以防止未授权的索引改动对系统至关重要。确保只有授权用户或程序才能执行新增、读取、更新和删除操作，这有助于确保文档库的完整性。

监控和维护

定期监控索引器的健康状况和性能至关重要。为了确保系统随时间顺畅运行，需要有健全的监控和维护程序来发现问题，如索引失败、资源瓶颈或过期的索引。

这些都是一些挑战性的但众所周知的软件工程问题，通过遵循良好的软件设计实践可以解决。

数据存储

由于我们要处理多种数据类型，需要为它们各自提供专门的存储。理解每种存储类型和每个用例的不同考虑因素至关重要。

嵌入式向量

数据库类型：SQL/NoSQL

单独存储文档的嵌入式向量可以实现快速重建索引，而无需重新计算整个文档库的嵌入式向量。另外，存储的嵌入式向量还可以作为备份，以确保在系统故障或更新情况下关键信息的保存。

文档

数据库类型：NoSQL

以原始格式存储文档对于持久化非常重要。这种原始格式构成了索引、解析和检索等多个处理阶段的基础，并且为未来系统的增强提供了灵活性，使得原始文档得以保存，并可根据需要重新处理。

聊天历史

数据库类型：NoSQL

存储聊天历史对支持 RAG 系统的对话功能至关重要。聊天历史存储使得系统能够记住用户先前的查询、回复和偏好，便于根据用户的独特背景来调整和定制未来的互动。这些历史数据对于利用研究来改进机器学习系统是非常有价值的。

用户反馈

数据库类型：NoSQL/SQL

用户反馈是通过 RAG 应用内的多种互动机制来系统性地收集的。在大多数大语言模型系统中，用户可以利用点赞/点踩、星级评分和文字反馈提供反馈。这些用户反馈汇集成一个珍贵的信息库，它包裹着用户的体验和感知，为系统的持续改进打下基础。

向量数据库

向量数据库是 RAG 强大的语义搜索能力背后的关键检索组件。然而，合适地选择这个组件至关重要，以避免潜在的问题。在选择过程中，需要考虑若干个[向量数据库的因素](#)。现在让我们来讨论其中一些。

召回率与响应时间

在向量数据库中，优化召回率（即相关结果的百分比）与优化响应时间（即返回结果所需的时间）之间需要做出权衡。如 [Flat](#)、[HNSW](#)（分层导航小世界）、[PQ](#)（产品量化）、[ANNOY](#) 和 [DiskANN](#) 等不同的索引之间在速度和召回率上做出了不同的折中。在您的数据和查询中进行基准测试，以便做出明智的选择。

成本

云原生的数据库通常根据数据存储和查询量来进行计费。对于数据量大的组织来说，这种模型能够避免基础设施成本。关键点包括，评估数据集的增长、团队的能力、数据敏感性，以及理解托管云解决方案的成本含义。

另一方面，自行托管为组织提供了更多地控制其基础设施的能力，并可能降低成本。然而，这也伴随着管理和维护基础设施的责任，包括可扩展性、安全性和更新等方面的考虑。

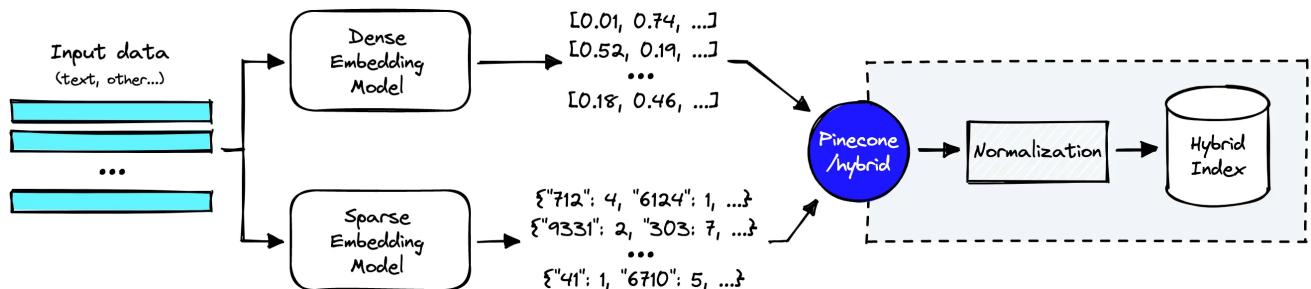
插入速度与查询速度

平衡插入速度与查询速度非常重要。寻找能够处理高插入速度需求的流式使用案例的服务提供商。然而，对大多数组织来说，将查询速度放在优先考虑的位置通常更加相关。评估在高负荷时的向量插入速度和查询延迟，以做出明智的决定。

内存索引与磁盘索引存储

在内存存储和磁盘存储之间进行选择，涉及到速度与成本的权衡。虽然内存存储提供更高的速度，但有些使用案例又需要存储大于内存容量的向量。像内存映射文件这样的技术允许在不牺牲搜索速度的情况下扩展向量存储。像 DiskANN 中的 Vamana 这样的新索引承诺提供高效的超出内存的索引。

全文搜索与混合向量搜索



图片来源：<https://www.pinecone.io/learn/hybrid-search-intro/>

单一的向量搜索可能不适合企业级应用程序。另一方面，混合搜索整合了密集型和稀疏型方法，但需要更多的努力来实施。通常情况下，需要实施密集型向量索引、稀疏型倒排索引和重排序步骤。在 [Pinecone](#)、[Weaviate](#) 以及 [Elasticsearch](#) 中，可以通过一个称为 alpha 的参数来调整密集型和稀疏型元素间的平衡。

过滤

现实世界中的搜索查询通常涉及对元数据属性的过滤。虽然预过滤搜索看起来是自然的选择，但可能会错过一些相关结果。如果过滤属性只占数据集的一小部分，后过滤搜索可能出现问题。自定义过滤搜索，如 [Weaviate](#) 所提供的，结合了预过滤和有效的语义搜索，并利用倒排索引分片和HNSW索引分片。

提高检索效果的技巧

最新的研究显示，[大语言模型可能会因无关上下文分散注意力](#)，并且如果上下文量过大（检索到的topK文档数量很多），则可能由于大语言模型的注意力模式[错失某些上下文](#)。因此，利用相关且多样的文档来改善检索效果至关重要。让我们看看一些已经证明有效的提高检索效果的技巧。

假设文件嵌入 (HyDE)

我们可以利用[HyDE](#)技术解决检索表现不佳的问题，特别是在处理那些可能导致信息难以找到的短小或不匹配的查询时。HyDE采用了一种独特的方法，借助像 GPT 这样的模型创建假设性文档。这些假设性文档捕捉了一些关键模式，但可能包含错误或误导性的细节。然后，一个智能文本编码器将这份假设文档转换为向量嵌入表示。这种嵌入式表示有助于比查询的嵌入式表示更加精准地在文档集合中找到类似的真实文档。

实验表明，HyDE比其他进阶方法的效果要好，这使其成为提升 RAG 系统性能的有力工具。

查询路由

当处理多个索引时，查询路由特别有用，它将查询引导到最相关的索引中，从而高效检索。这种方法通过确保每个查询都被正确地引导到合适的索引，优化了信息检索的准确性和速度。

在企业搜索的背景下，由于数据来自不同的来源，如技术文件、产品文档、任务和代码库，查询路由成为一种强大的工具。比如，如果用户正在寻找与特定产品特性相关的信息，查询路由可以智能地将其引导到包含产品文档的索引，增强搜索结果的精度。

重排序器

当编码器的检索结果未能提供最佳质量时，可以使用[重排序器](#)来增强文档排序效果。在交叉编码器设置中使用像[BGE-large](#)这样的开源仅编码器变压器已成为常规实践。近来，像[RankVicuna](#)、[RankGPT](#)和[RankZephyr](#)这样的仅解码器方法进一步提升了重排序器的表现。

引入重排序器有其优点，它能减少大语言模型的幻象现象，并改善系统在领域外的泛化能力。但也有不足之处，复杂的重排序器会因为计算负担增加而引入延迟，这会影响实时间应用的响应速度。此外，部署先进的重排序器可能会耗费大量资源，需仔细权衡性能提升与资源使用之间的平衡。

最大边缘相关性（MMR）

MMR 是一种旨在提高对查询响应中的检索项多样性的方法，以避免冗余。MMR 不仅仅专注于检索最相关的条目，而是力求在相关性和多样性之间找到平衡。它就像在派对上为一个朋友介绍人。起初，它会根据朋友的喜好确定最相配的人。然后，它会寻找一个略有不同的人。这个过程持续进行，直到达到所需的介绍次数。MMR 确保呈现的条目集合更加多样化和相关，最小化了重复性。



原始的MMR

自动裁剪

Weaviate 的自动裁剪功能限制了返回搜索结果的数量，它通过探测分数相近的对象群组实现。它通过分析搜索结果的分数来工作，并识别这些值之间的显著变化，这可能表明结果从高相关性过渡到较低相关性。

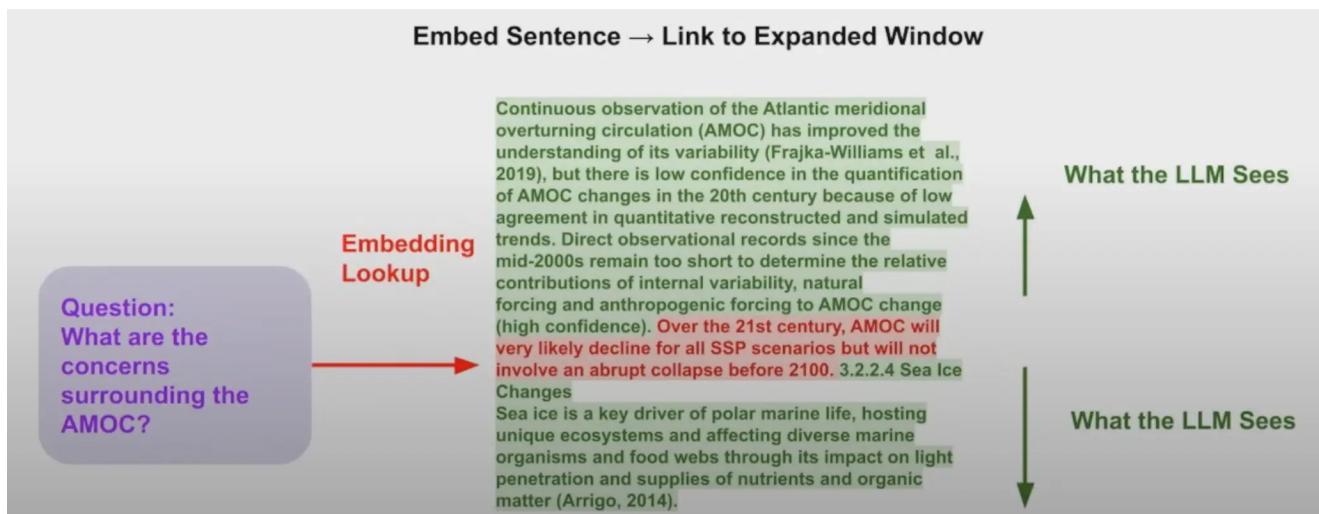
例如，考虑一个返回以下距离值的搜索：

[0.1899, 0.1901, 0.191, 0.21, 0.215, 0.23]。

自动裁剪将返回以下结果：

- autocut: 1: [0.1899, 0.1901, 0.191]
- autocut: 2: [0.1899, 0.1901, 0.191, 0.21, 0.215]
- autocut: 3: [0.1899, 0.1901, 0.191, 0.21, 0.215, 0.23]

递归检索



图片来源：<https://youtu.be/TRjq7t2Ms5I?si=D0z5sHKW4SMqMgSG&t=742>

递归检索也被称为从小到大的检索技术，在检索时嵌入较小的文本块，并返回用于语言模型合成的更大的父级上下文。较小的文本块有助于提高检索的准确性，同时更大的块为语言模型提供了更丰富的上下文信息。这一连贯的过程通过首先关注更小、信息更密集的单位来优化检索的准确性，然后有效地将它们与它们更广泛的背景父块连接起来进行合成。

窗口句子检索

检索过程会获取单句并返回围绕该特定句子的文本窗口。窗口句子检索确保获得的信息不仅准确，而且与上下文相关，提供了关于主要句子的全面信息。

生成器

在我们讨论完所有的检索组件之后，让我们来讨论一下生成器。在自托管推断部署和私有API服务之间需要进行慎重的考虑和权衡。这本身就是一个相当大的话题，为了不让你感到不知所措，我将简要介绍一下。

API 的关键考虑

在评估大语言模型的 API 服务器时，我们必须优先考虑确保无缝整合和强大性能的功能。一个设计得当的 API 应该能像使用普通发射器一样方便地启动流行的大语言模型，并且还要处理好生产环境准备、安全性以及[幻象检测](#)等关键问题。比如，[HuggingFace 的 TGI 服务器](#)就体现了一整套这些原则的功能。让我们来了解一下大语言模型服务器需要的一些最受欢迎的功能。

性能

一个高效的 API 必须以性能为首要任务，以满足不同用户的需求。张量并行作为一项特殊功能，它通过在多个GPU上进行更快的推理处理，提高了整体加工速度。此外，连续批量处理传入请求确保了总吞吐量的提升，使系统更为响应和可扩展。量化特性的引入，特别是与比特和字节以及 GPT-Q 相关的量化，进一步优化了 API 的效率，以适应各种使用场景。利用优化的变压器代码的能力，可确保在最受欢迎的架构上顺畅地进行推理。

生成质量增强功能

为了提高生成的质量，API 应该包括可以改变输出的功能。概率处理器包括温度缩放、top-p、top-k 和重复惩罚，允许用户根据自己的偏好调整输出。此外，停止序列提供了对生成过程的控制，使用户能够管理和优化内容生成过程。用于幻象检测的对数概率，是确保生成输出与预期上下文一致并避免误导信息的额外精炼层。

安全性

在处理大语言模型和企业应用案例时，API 的安全性至关重要。安全地加载权重的 Safetensors 是一项关键功能，通过防止对模型参数未授权的篡改，有助于安全部署模型。此外，水印技术的引入增加了额外的安全层，使得大语言模型的使用能够被追踪和负责。

用户体验

在用户体验方面，令牌流式传输作为关键功能浮现出来，它有助于实现无缝互动。使用服务器发送事件（SSE）进行令牌流传输，提高了 API 的实时响应能力，为用户提供了更流畅、更互动的体验。这确保用户可以逐渐接收生成的内容，从而改善了大语言模型的整体参与度和可用性。

自托管推理

自托管推理涉及到在 AWS、GCP 或 Azure 等云服务提供商提供的服务器上部署大语言模型。选择诸如 TGI、Ray 或 FastAPI 等服务器是一个直接影响系统性能和成本的关键决策。考虑的因素包括计算效率、部署的简易性以及与选定的大语言模型的兼容性。衡量大语言模型推理性能至关重要，像[Anyscale's LLMPERF Leaderboard](#)这样的排行榜是非常有价值的。它根据关键性能指标对推理提供者进行排名，包括首令牌时间（TTFT）、令牌间延迟（ITL）和成功率。负载测试和正确性测试对评估托管模型的不同特性至关重要。

在新方法中，[Predibase 的 LoRAX](#) 推出了一种高效服务多个精调大语言模型的创新方式。它解决了使用共享 GPU 资源为多个精调模型服务的挑战。

私有 API 服务

OpenAI、Fireworks、Anyscale、Replicate、Mistral、Perplexity 和 Together 等公司提供的大语言模型 API 服务呈现了替代的部署方法。了解它们的特性、定价模型和[大语言模型性能指标](#)至关重要。例如，OpenAI 的基于令牌的定价模型，对输入令牌和输出令牌区分开来，这可能会显著影响使用 API 的总成本。在比较私有 API 服务与自托管大语言模型的成本时，你必须考虑 GPU 成本、使用率和可扩展性问题。对某些人来说，请求速率的限制可能是限制性的因素（哈！）。

提高 RAG 的提示技术

有多种提示技术用于提高 RAG 的输出效果。在我们[精通 RAG 系列的第二部分](#)中，我们深入探讨了五种最有效的技术。其中许多新技术的性能超过了思维链（Chain-of-Thought）。你也可以结合使用这些技术来尽量减少幻象。

LLM Prompting Techniques For RAG

Name	How it works?	Ease of implementation	Increase of input token	Increase of output token
Thread of Thought (ToT)	Break down and analyzes extensive contexts for selecting relevant information	Easy	Yes	Yes
Chain of Note (CoN)	Generate sequential reading notes for retrieved documents → evaluate their relevance to the given question → integrate information to formulate the final answer	Easy	Yes	Yes
Chain of Verification (CoV)	Draft a response → plan verification questions → answer those questions independently → generate final verified response	Hard	Yes	Yes
EmotionPrompt	Add an emotional prompt to the original prompt	Easy	Yes	No
ExpertPrompting	Add synthesized expert background generated with another few shot prompt	Easy	Yes	No



大语言模型的 RAG 提示技术

输出安全保护

输出安全保护的功能类似于输入安全保护，但专门用于检测生成输出中的问题。它重点检测幻象、竞争对手的提及以及潜在的品牌损害，这是[RAG 评估](#)过程的一部分。其目标是阻止生成不准确或在伦理上存疑的信息，这些信息可能与品牌价值观不符。通过积极监视和分析生成的内容，输出安全保护确保生成的内容在事实上保持准确，伦理上可靠，并与品牌准则一致。

这里有一个可能损害企业品牌的响应实例，但若有合适的输出安全保护将会被阻止：

Query

What are your thoughts on climate change?

Correct Response

Climate change is a complex global issue with significant environmental impacts. It requires collaborative efforts to address and mitigate its effects.

Incorrect Response

Climate change is just a bunch of hype created by fearmongers. People need to focus on real issues instead of getting worked up over exaggerated threats.

 Galileo

有害输出案例

用户反馈

当生成的输出交付给用户后，获得用户的正面或负面反馈是十分有益的。用户反馈对于提高RAG系统的持续改进非常有帮助，因为这是一个持续的过程，而非一次性的任务。这其中不仅涉及执行自动化任务，如重新索引和重复实验，而且还需要系统地融合用户的见解来大幅增强系统。

系统改善中最有效的手段在于积极修复基础数据中的问题。RAG系统应包括迭代性的工作流来处理用户反馈并推动持续改进。

用户互动与反馈收集

用户与RAG应用交互，并使用点赞/点踩或星级评分等功能来提供反馈。这种多样的反馈方式为收集用户的体验和对系统性能的看法提供了宝贵的资源。

问题识别与诊断检测

搜集反馈之后，团队可以进行全面的分析来确定哪些查询的性能可能不佳。此过程包括审查检索到的资源以及深入研究以确定性能问题是来自检索、生成还是底层数据源。

数据改进策略

一旦识别出问题，尤其是那些根植于数据本身的问题，团队可以采取战略行动来提升数据品质。这可能包括更正不完整信息或重组组织混乱的内容。

评估与测试协议

在实施数据改进后，系统必须对先前性能不佳的查询进行严格评估。从这些评估中获得的见解可以被有序地融入测试套件中，确保基于实际交互持续检查和改进。

通过在这一全面的反馈循环中积极参与用户，RAG 系统不仅解决了通过自动化过程发现的问题，还利用了用户体验的丰富性。

监控性

建立 RAG 系统并不是随着系统部署到生产环境就结束了。即便配备了强固的安全措施和高质量数据用于[微调](#)，模型一旦部署到生产中，也需要持续的监测。生成式AI应用除了标准指标，如延迟和成本外，还需要特定的[大语言模型监测性](#)功能来检测和纠正像幻觉、越界查询和链路故障等问题。下面是大语言模型监测的几个关键点。

提示分析及优化

识别与提示相关的问题，并利用实时生产数据迭代，使用健全的评估机制来识别和解决诸如幻觉等问题。

大语言模型应用的可追溯性

捕捉像 Langchain 和 LlamaIndex 这样的通用框架产生的大语言模型轨迹，以调试提示语和步骤。

信息检索优化

对 RAG 参数进行故障诊断和评估，以优化对大语言模型性能至关重要的检索过程。

警报

在系统行为偏离预期时接收警报，例如错误增加、延迟增高和幻觉发生。

首先，实时监控应用程序在生产环境中的性能、行为和整体状况至关重要。密切注意SLA的遵守情况，并设置警报机制，迅速处理任何偏离。通过分析使用模式和资源消耗来跟踪运行大语言模型应用的成本，辅助你进行成本优化。

Galileo推出了[大语言模型工作室](#)，专门为大语言模型监测设计，可在用户投诉前即时发出警告并采取纠正措施。Galileo的安全保护指标旨在监督模型的质量和安全性，涵盖了实在性、不确定性、事实性、语调、毒性、个人身份信息等方面。这些在评估和测试阶段曾使用的指标现可无缝融入监测环节。

另外，你可以根据具体需要定制监控过程中的自定义指标。利用监视数据生成的洞察和警报，及时掌握潜在问题、异常情况或需要关注的改进领域。这种全面的方法确保了你的大语言模型应用在现实世界场景中能够高效且安全运行。

缓存

对于那些大规模运营的公司来说，成本可能会成为一个限制因素。在这种情况下，缓存是一种极好的节约成本的方法。缓存的工作原理是将提示和相应的回复储存在数据库中，以便未来可以快捷地提取使用。这种策略性的缓存机制赋予了大语言模型应用三个明显的优势，它既加快了响应速度，又节约了成本。

提升生产推理的效率

在生产环节，缓存能够加速推理过程，使其更便宜且更高效。某些查询可以利用缓存的响应几乎达到零延迟，优化了用户体验。

缩短开发周期

在开发阶段，由于无需重复调用 API 来处理相同的提示，缓存可以显著减少开发所需时间，降低开发成本。

数据存储

拥有存储全部提示的综合数据库简化了大语言模型的微调流程。利用存储的提示与回复对，可以根据积累的数据更加高效地优化模型。

如果你确实需要，可以采用 [GPTCache](#) 来实施缓存，包括精确匹配和类似匹配。它为缓存的性能提供了如缓存命中率、延迟和准确性等关键指标，这些指标有助于持续优化，确保高效运行。

多租户模式

SaaS 软件通常需要服务多个租户，既要保证简洁性，又要注重隐私保护。在 RAG 系统中实现多租户模式的目标是建立一个既有效获取信息又尊重每位用户数据隐私的系统。简而言之，每位用户与系统的互动都是独立的，从而确保系统仅查看和利用针对该用户的信息。

一种简单建立多租户模式的方法是使用元数据。向系统添加文档时，我们会在元数据中包括特定的用户信息。这样，每个文档都与特定用户关联。当用户进行搜索时，系统会使用这些元数据进行筛选，只显示与该用户相关的文档。然后通过智能搜索为用户找到最重要的信息。这种做法避免了不同用户之间的私密信息混淆，保护了每个人的数据安全和私密性。

了解如何利用 [Llamaindex](#) 实现多租户的操作方法。

结论

显而易见，构建一个健壮、可扩展的企业级 RAG 系统需要精心安排多个互连的组件。从用户认证到输入安全保护，再到查询重写、编码、文档插入以及像向量数据库和生成器这样的检索组件，每一步都在系统性能构建中发挥着关键作用。

在不断演进的 RAG 系统领域，我们希望这份实用的指南能够为开发者和领导者提供实践性的洞察，助力他们取得实际成效！