# RAG 串讲

MLE 算法指北

2025 年 3 月 14 日

## 1 RAG 概述

### 1.1 LLM 的缺陷

- 幻觉（Hallucination） LLM 可能生成 ** 无事实依据、不与现实世界一致 **，甚至 ** 完全虚构 ** 的内容。这导致模型在信息生成过程中可能产生错误。
- 知识更新滞后（Knowledge Cut-off） LLM ** 知识的有效性取决于训练数据的时间 **，因此存在知识更新滞后的问题。
- 回答缺乏透明度（Lack of Transparency in Answers）LLM 生成的回答通常 ** 缺乏引用来源 **，加上幻觉问题，导致用户难以判断答案的真实性，降低了模型的可信度。

为了缓解这些问题，**RAG（Retrieval-Augmented Generation）** 提出了一种解决方案，即：

- 在 LLM 生成答案时 ** 检索外部数据 **
- 通过提供 ** 事实性上下文 ** 减少幻觉问题
- 提高回答的 ** 可解释性和用户信任度 **

### 1.2 定义（什么是 RAG）

**RAG（Retrieval-Augmented Generation）** 是一种结合 ** 检索（Retrieval）+ 生成（Generation）** 的 AI 方案，旨在提高 LLM 生成的准确性。其核心原理是：

1. ** 检索阶段（Retrieval）**：
   - 用户输入查询，系统从外部知识库（如 Wikipedia、企业文档）检索相关信息
   - 采用 ** 向量搜索（Vector Search）** 或 **BM25 关键词匹配 **
2. ** 生成阶段（Generation）**：
   - LLM 结合 ** 检索到的信息 ** 和 ** 自身的知识 ** 进行回答
   - 通过 **Prompt Engineering** 让模型 ** 优先引用检索结果 **

### 1.3 RAG 主要组件

RAG 体系结构通常包括以下核心组件：

- **向量数据库（Vector DB）**：用于存储和检索嵌入向量，例如 FAISS、Pinecone、Weaviate。
- **嵌入模型（Embedding Model）**：将文本转换为向量，如 OpenAI Embeddings、BERT。
- **检索模型（Retriever）**：搜索相关信息，可使用 **BM25、Dense Retrieval（DPR）** 等方法。
- **LLM（Large Language Model）**：如 GPT-4，负责生成最终答案。
- **融合策略（Fusion Strategy）**：结合检索信息 + LLM 结果，提高生成的可信度。

### 1.4 RAG 的优势

相较于单纯的 LLM，RAG 方案具有以下优势：

| ** 特性 ** | **RAG 优势 ** |
|---|---|
| 事实准确性 | 通过外部检索，减少幻觉问题 |
| 知识更新 | 依赖实时数据，无需重新训练模型 |
| 可解释性 | 生成结果带有检索来源，提高可信度 |
| 数据可控性 | 可使用特定数据源，如企业知识库 |

表 1: RAG 方案的主要优势

### 1.5 RAG 的应用场景

- **企业知识问答（Enterprise Q&A）**：
  - 结合企业内部文档，实现精准问答
  - 例如 Microsoft Copilot、企业搜索
- **法律/医疗领域问答**：
  - 结合权威法律法规、医学文献，提高回答准确性
  - 例如 LexisNexis AI、医学 AI 诊断
- **金融风控**：
  - 结合市场数据和金融法规，辅助投资分析
  - 例如 BloombergGPT
- **搜索增强对话（Chatbot Augmentation）**：
  - 结合搜索引擎和 LLM，提高问答质量
  - 例如 Perplexity AI、ChatGPT Bing

## 2 RAG 技术路线

按照 RAG 的发展演进路线，其架构可以分为以下三类：

- **简单 RAG（Naive RAG）**
- **高级 RAG（Advanced RAG）**
- **模块化 RAG（Modular RAG）**

### 2.1 简单 RAG（Naive RAG）

**Naive RAG** 是最基础的 RAG 版本，其工作流程如下：

1. **用户查询（User Query）**：用户输入问题。
2. **文档切分（Document Chunking）**：将知识库中的文档分成小段。
3. **向量检索（Vector Retrieval）**：使用向量数据库（如 FAISS、Pinecone）检索相关文档。
4. **相关文档匹配（Related Document Chunks）**：获取最相关的文本片段。
5. **Prompt 构造**：将检索到的文本与用户查询一起输入 LLM。
6. **LLM 生成回答**。

尽管 Naive RAG 在简单场景下表现良好，但其 ** 检索质量和生成能力较弱 **，可能导致：

- **检索到的文档片段不够精准**，导致生成的答案仍然出现幻觉。
- **缺乏对检索结果的过滤与优化**，可能包含不相关信息。

### 2.2 高级 RAG（Advanced RAG）

**Advanced RAG** 通过 ** 优化检索阶段（Pre-Retrieval）和生成阶段（Post-Retrieval）** 来提高回答质量，其优化方法包括：

- **检索前优化（Pre-Retrieval）**：
  - 文档数据清理（Data Cleaning）
  - 文本去重（Deduplication）
  - 向量索引优化（Efficient Indexing）
- **检索后优化（Post-Retrieval）**：
  - 文档 ** 重排序（Rerank）**，提高相关性
  - ** 噪声过滤（Filter）**，去除无关内容
  - ** 提示词优化（Prompt Compression）**，减少冗余信息

### 2.3 模块化 RAG（Modular RAG）

**Modular RAG** 进一步引入 ** 多种可插拔组件 **，使系统更加灵活和可扩展，包含：

- **检索模块（Retrieve）**：支持多个检索方式，如密集检索（DPR）和 BM25 结合。
- **重新排序（Rerank）**：对检索结果进行排序，提高相关性。
- **过滤（Filter）**：剔除无关或低质量的检索结果。
- **阅读理解（Read）**：结合 LLM 对检索到的信息进行摘要或解释。
- **反思（Reflect）**：使用额外推理模块优化 LLM 生成结果。
- **批判性思考（Criticize）**：利用额外模型检查 LLM 的输出是否符合事实。

### 2.4 RAG 方案对比

| ** 特性 ** | **Naive RAG** | **Advanced RAG** | **Modular RAG** |
|---|---|---|---|
| 检索优化 | 无优化 | 增加去重、索引优化 | 多级检索，支持混合检索 |
| 结果排序 | 无排序 | 简单 rerank | 支持深度 rerank |
| 过滤能力 | 无 | 低级过滤 | 复杂过滤规则 |
| 生成优化 | 无 | 提示词优化 | 结合反思（Reflection）、批判（Criticize） |

表 2: 不同 RAG 方案对比

## 3 RAG 组件详细介绍

### 3.1 数据清洗（Data Cleaning）

数据清洗是确保 ** 高质量知识库 ** 的第一步，影响检索与 LLM 生成效果。主要包括：

- **去重（Deduplication）**：去除重复文档或内容，提高存储效率，减少无效检索。
- **格式标准化（Normalization）**：统一 ** 文本编码、标点符号、日期格式 **，保证数据一致性。
- **噪声过滤（Noise Filtering）**：删除 ** 无关内容、广告、低质量文本 **，避免干扰 LLM 生成。
- **停用词处理（Stopwords Removal）**：去除对语义无贡献的停用词，如”the”、”is”。
- **语言检测（Language Detection）**：确保 LLM 仅接收目标语言文本，提高回答质量。

—

### 3.2 数据分片 Chunking

Chunking 主要有 ** 基于规则 ** 和 ** 基于模型 ** 两种策略。

### 3.2.1 基于规则的 Chunking

- **固定长度分片（Fixed-Length Chunking）**：
  - 按 ** 固定字符数 ** 或 ** 固定单词数 ** 切分，如每 512 词分为一个 Chunk。
  - ** 优点 **：简单易实现，适合结构化文本（如 Wikipedia）。
  - ** 缺点 **：可能导致句子或段落被拆散，影响语义完整性。
- **基于段落的 Chunking（Semantic Chunking）**：
  - 以 ** 自然语言的逻辑单元 **（如句子、段落）进行分片，保持上下文完整性。
  - ** 优点 **：减少语义破坏，适用于法律、医学文档。
  - ** 缺点 **：Chunk 长度不均匀，可能影响索引效率。
- **滑动窗口分片（Sliding Window Chunking）**：
  - 相邻 Chunk 之间有 ** 部分重叠 **，如窗口大小 256 词，滑动步长 128 词。
  - ** 优点 **：保证跨段落上下文信息，减少查询时的信息丢失。
  - ** 缺点 **：增加存储和计算开销。

### 3.2.2 基于模型的 Chunking

基于 **NLP 预训练模型 ** 进行智能 Chunking，可以更好地 ** 保持语义完整性 **，包括：

- **Transformer 句子分割（Sentence Segmentation with Transformers）**：
  - 使用 Transformer 模型（如 BERT、RoBERTa）检测 ** 句子边界 **，以 ** 高语义相关性 ** 进行分片。例如 BERT 的 NSP 预训练任务。
  - 适用于 ** 复杂文档，如法律、医学文本 **，避免破坏逻辑结构。
- ** 基于依存关系的 Chunking（Dependency Parsing Chunking）**：
  - 使用 ** 依存句法分析（Dependency Parsing）** 识别 ** 主谓宾结构 **，以语法结构为基础拆分 Chunk。
  - 适用于 ** 技术文档 **，如 API 说明书、论文摘要等。
- ** 基于 LLM 的 Chunking（LLM-Based Chunking）**：
  - 让 LLM ** 自适应判断 Chunk 切分点 **，例如：
    ”根据上下文，哪些部分应该作为独立片段？”
  - 适用于 ** 非结构化文本 **（如用户评论、社交媒体内容）。

### 3.2.3 Small2Big Chunk 策略

**S**mall2Big Chunking 策略的核心思想是：

先生成 ** 较小的 Chunk**，再根据 ** 语义相似度 ** 合并成 ** 较大的 Chunk**，最终形成更合理的文档分片结构。

该方法 ** 结合了固定长度切分与语义分析 **，避免了传统 Chunking 方法的弊端：- **过小 Chunk** 可能丢失上下文信息，导致 LLM 生成错误答案。- **过大 Chunk** 可能增加检索噪声，影响搜索精准度。

Small2Big Chunking 主要包含以下步骤：

1. ** 初步切分小片段（Small Chunks）**
   - 使用 ** 固定长度切分 **（如每 100 词一个 Chunk）。
   - 或者采用 ** 自然段落切分 **，保持文本语义完整性。
2. ** 计算 Chunk 之间的语义相似度 **
   - 使用 **BERT/SBERT 向量嵌入 ** 将 Chunk 表示为向量：

$$v_i = \text{Embedding}(C_i) \tag{1}$$

   - 计算相邻 Chunk 之间的 ** 余弦相似度 **：

$$S_{i,i+1} = \frac{v_i \cdot v_{i+1}}{\|v_i\| \|v_{i+1}\|} \tag{2}$$

- 设定相似度阈值 $\tau$，如果 $S_{i,i+1} > \tau$，则合并 Chunk。

3. ** 合并相似 Chunk，形成较大 Chunk（Big Chunks）**
   - 相似度高的 Chunk 进行合并，形成更大、更有信息密度的片段：
   $$C_i' = C_i \cup C_{i+1}, \quad \text{if } S_{i,i+1} > \tau \tag{3}$$
   - 逐步执行，直到不再满足合并条件。
4. ** 存入向量数据库 **
   - 采用 FAISS/Pinecone 存储优化后的 Chunk。

## 3.3 Embedding 计算

Embedding 是 **RAG 检索效率的核心**。不同任务需要不同的 Embedding 模型，常见选择包括：

- **开源 Embedding 模型**
  - **OpenAI Embeddings**：如 text-embedding-ada-002，支持 1536 维向量，适用于通用 NLP 任务。
  - **BERT-based Embeddings**：如 SBERT，适用于语义匹配和问答任务。
  - **MTEB（Massive Text Embedding Benchmark）排名靠前的模型**，例如 BAAI/bge-base-zh，Alibaba-NLP/GTE-base。
- **自定义 Finetuned Embedding**
  - 适用于法律、医学等特定领域，通过微调现有 Embedding 模型适应专业术语。
  - 如对医学文本进行自监督训练（Contrastive Learning），提升向量匹配效果。

### 3.3.1 Query embeding 优化与 doc embeding 优化

Query 质量决定了检索的准确性，因此需要优化用户查询，提高召回率。**Query 改写（Query Rewriting）**

- 使用 LLM 对 Query 进行 ** 重写 **，生成更自然、更易匹配的表达方式。

**HyDE（Hypothetical Document Embeddings）** **HyDE（Hypothetical Document Embeddings）** 通过 ** 生成 Hypothetical Documents** 提高检索效果：

- **原理**：
  - LLM ** 先生成一个 Hypothetical Response** 作为 Query 的潜在答案。
  - 计算该 ** 假设文档的 Embedding**，再进行检索。

除了 Query 端优化，文档（Doc）端也可以进行预处理，以提高检索质量。**假设问题（Hypothetical Question）优化**

- 让 LLM ** 自动生成 ** 每个文档可能对应的 ** 假设问题（Hypothetical Questions）**。
- 生成的问题 + 文档的 Embedding 一起存储，提高召回率。

**摘要 Embedding**

- 对每个 Doc 生成摘要，并将摘要的 Embedding 存入索引。
- 适用于 ** 长文档（如论文、专利）**。

## 3.4 检索 Retrieval 算法

RAG（Retrieval-Augmented Generation）中的 Retrieval 算法是系统精准高效检索内容的核心技术，具体分为以下几类：

### 3.4.1 关键词检索（Keyword-based Retrieval）

关键词检索以明确的关键词为基础，典型算法为 BM25。
BM25 算法公式如下：

$$\text{BM25}(Q, D) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \left(1 - b + b \frac{|D|}{avgdl}\right)} \tag{4}$$

#### 3.4.2 语义检索（Semantic Retrieval）

语义检索利用向量空间表示文本语义，通过向量之间的相似性度量进行检索。
常用的算法包括：

- **余弦相似度（Cosine Similarity）**

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{5}$$

- 近似最近邻搜索（Approximate Nearest Neighbors，ANN）算法，如 FAISS、HNSW。
  - FAISS：适合大规模文档的高效向量检索。
  - HNSW：图结构算法，实时检索效率高。

#### 3.4.3 混合检索（Hybrid Retrieval）

混合检索结合了关键词检索和语义检索两种方式，常见策略包括：

- 并行召回：同时使用关键词与语义向量搜索，合并结果。
- 串行召回：关键词召回后，再进行语义二次精排。

### 3.5 RAG 中的重排序（Rerank）算法

RAG 检索流程通常包括两个阶段：

1. **召回（Retrieval）**：基于关键词（如 BM25）或语义（Embedding ANN）快速检索出候选文档。
2. **重排序（Rerank）**：对初步召回的文档进行更为精细的语义相关度计算，得到精确的排序结果。

重排序算法负责第二阶段的任务，进一步提高检索精确性和召回效果。

#### 3.5.1 重排序算法流程

Rerank 算法一般包含如下流程：

- **输入**：候选文档集合 $\{D_1, D_2, ..., D_n\}$ 和查询文本 $Q$；
- **精细打分**：利用高精度的 Cross-Encoder 模型分别计算每对 $(Q, D_i)$ 的相关度评分；
- **重排序**：按照分数高低重新排序，取前 $k$ 个文档作为最终结果。

#### 3.5.2 Cross-encoder 算法

Cross-Encoder 模型将查询与文档一起输入，直接输出相关性得分：

$$Input = [CLS]\, Q\, [SEP]\, D_i\, [SEP] \tag{6}$$

模型输出一个标量得分，代表查询与文档的匹配程度，公式表示为：

$$Score(Q, D) = f_{\text{CrossEncoder}}([CLS]\ Q\ [SEP]\ D\ [SEP])$$

常见 Cross-Encoder 模型包括：

- 基于 BERT 的 Cross-Encoder；
- 领域特定 Finetuned Cross-Encoder。

#### 3.5.3 算法效果优化方法

提升 Rerank 算法效果的常见方法包括：

- **领域微调（Finetune）**：使用领域数据微调 Cross-Encoder 模型；
- **知识蒸馏（Knowledge Distillation）**：训练轻量级模型，降低延迟；
- **难负例（Hard Negative）训练**：增强模型识别能力。

## 3.6 生成策略（Generation）

生成策略是 RAG 系统中检索阶段完成后的下一步，用于指导生成模型（如 GPT 系列模型）根据检索结果生成精准且连贯的答案。合理选择生成策略可以显著提高答案的质量、准确性与相关性。

### 3.6.1 生成策略定义与流程

生成策略位于检索阶段之后，具体流程如下：

1. 检索阶段获取与查询相关的文档或片段；
2. 将检索内容以适当方式融入 Prompt 中；
3. 使用生成模型（如 GPT-4）基于 Prompt 生成最终答案。

### 3.6.2 Prompt-based 生成策略

Prompt-based 生成策略将检索到的文档直接放入 Prompt 中进行生成：

$$\text{Prompt} = [\text{Instruction}] + [\text{Retrieved Docs}] + [\text{Query}]$$

此方法简单高效，易于实现，但受限于模型最大上下文长度，且可能产生幻觉（Hallucination）问题。

### 3.6.3 融合生成策略（Fusion-in-Decoder）

融合生成策略通过在 Decoder 阶段同时结合多个检索文档的信息，增强生成答案的精准度：

- **RAG-Sequence**：逐步选择合适文档生成答案，每一步动态选择上下文。
- **RAG-Token**：每生成一个词（Token）都动态选择不同文档的信息以支持当前生成的内容。

融合生成能有效减少幻觉问题，但推理速度较慢，实现成本相对较高。

### 3.6.4 迭代生成策略（Iterative Generation）

迭代生成策略允许模型在多轮迭代中逐步优化答案：

- 根据初步答案再次检索相关文档；
- 重新生成答案，迭代直至达到期望质量；
- 常用于需要极高准确性的场景。

### 3.6.5 自一致性生成策略（Self-Consistency）

自一致性策略使用多次生成并融合多个候选答案以提高整体质量：

- 对同一检索结果多次调用生成模型，得到多个答案；
- 采用多数投票或评分融合机制选择最优答案；
- 能有效提高结果稳定性，但计算成本较高。

### 3.6.6 生成策略的优化方法

实际中，通过以下方式进一步优化生成效果：

- **Prompt 工程优化**：合理设计 Prompt 结构，提升生成质量；
- **领域特定模型微调（Finetune）**：在专业领域内优化生成效果；
- **Temperature 和 Top-k 调整**：调低 temperature 参数以提高生成稳定性和一致性。

## 3.7 增强策略（Augmentation Strategy）

在 RAG（Retrieval-Augmented Generation）系统中，增强策略（Augmentation Strategy）指的是如何高效地结合检索到的外部知识与生成模型自身的知识，以提升生成结果的相关性、准确性及可靠性。这一过程又被称为**增强（Augmentation）**或信息融合。

纯生成模型（如 GPT 模型）虽然强大，但面临幻觉问题（Hallucination）和知识过期问题（Knowledge Cut-off）。RAG 通过引入外部知识库增强生成质量，具体实现为：

1. 在生成前检索外部知识；
2. 将知识高效融入生成模型的上下文中；
3. 模型基于真实、准确的信息生成答案。

增强策略负责决定如何将检索到的信息更有效地融入生成过程，以提升生成结果的准确性和可信性。

### 3.7.1 常见增强策略介绍

**（1）文本拼接增强（Concatenation-based）** 最简单的增强方法，将检索到的内容直接拼接到 Prompt 中：

$$\text{Prompt} = \text{Context}_1 + \text{Context}_2 + \cdots + \text{Context}_k + \text{Query}$$

**优点：**简单、实现便捷。**缺点**：

- Prompt 长度有限，受限于模型 Token 上限；
- 可能包含过多无关信息，导致模型注意力分散。

**(2)上下文融合增强(Fusion-in-Decoder）** 上下文融合增强(FiD)通过编码多个文档信息,并在 Decoder 阶段进行动态注意力融合：

- 使用独立的 Encoder 对每个文档编码；
- Decoder 阶段通过 Cross-Attention 同时关注多个文档，实现更细粒度的信息融合。

**（3）迭代增强（Iterative Augmentation）** 迭代增强方法通过多轮检索与生成相结合：

1. 首轮检索和生成初步答案；
2. 根据答案再次检索补充或验证信息；
3. 通过迭代式改进答案质量，降低幻觉风险。

该方法适合精确要求高的领域，例如医疗、法律等场景。

**（4）自监督一致性增强（Self-Consistency）** 自一致性增强基于多次独立生成，融合多个答案的策略：

- 针对同一检索内容，调用模型多次生成不同答案；
- 利用多数投票或概率加权选出最优答案；
- 提高生成内容稳定性和准确性。

### 3.7.2 增强策略的优化建议

为提升增强效果，实践中可采取以下优化策略：

- **Prompt 工程优化**：
  - 精炼 Prompt 格式，仅保留最相关内容；
  - 提高模型注意力和信息利用效率。
- **文档片段精细化**：
  - 使用 Rerank 算法筛选文档质量，降低干扰信息；
  - 提升增强文档片段的有效性和精确性。
- **领域微调增强模型**：
  - 针对垂直领域数据微调生成模型，进一步提高准确性；
  - 有效减少幻觉问题。

# 4 RAG 系统的评估方法与指标

为了确保检索增强生成（RAG）系统的有效性，评估环节至关重要。RAG 评估需要综合考量检索质量和生成质量，因此评估方法与指标通常较为复杂。本节详细介绍 RAG 的常用评估方法、主流评估框架以及具体的评估指标。

## 4.1 评估方法（Evaluation Methods）

RAG 系统评估一般涵盖两个层面：

1. **检索评估（Retrieval Evaluation）**：评估检索召回质量，即系统能否召回正确且相关的文档。
2. **生成评估（Generation Evaluation）**：评估最终生成答案的质量，包括准确性、流畅性与一致性。

**检索评估方法**

检索评估关注检索阶段的准确性和召回率，常用方法包括：MRR, Recall@k, NDCG, Hit rate 等等，主要关注的是 context relevance 指标，及检索到的 chunks 与 query 的相似度。

**生成评估方法**

生成评估可以从两个角度出发，一种衡量生成模型给出的答案能否忠实地反映检索到的上下文内容，避免产生幻觉或无根据的答案。这一维度确保生成的答案是可信的和基于事实的。另一种关注模型所生成的答案是否直接有效地回答了用户问题，体现了答案的准确性和有效性。

## 4.2 主流评估框架（Evaluation Frameworks）

常见的 RAG 评估框架包括：

- **LangChain Evaluation**：内置标准化评估流程，支持检索评估、生成评估及自动指标计算。
- **LlamaIndex Eval Framework**：专注于 RAG 场景，提供综合的检索与生成质量评估。
- **Haystack Eval Framework**：集成了检索与问答场景的评估能力，适用于工业级 RAG 系统。
- **RAGAS（Retrieval-Augmented Generation Assessment）**：专门用于 RAG 场景的评估框架，可同时评估检索与生成质量。