

RAG Retrieval Evaluation & Failure Attribution

RAG 检索质量评估与失败归因——把「答错了」拆开来

2026年10月16日 | AI Agent 知识分享 | 第78期

一、一句话概括

检索质量评估把「RAG 答错了」从一句笼统的抱怨拆成可定位的故障：检索层看 hit rate / MRR / nDCG@10，生成层看 faithfulness / relevancy，再用七类失败点把每条错例归因到具体环节——不做这一步，优化就是在错的那一层烧钱。

二、核心原理

>> 一、为什么必须把检索和生成拆开测

一个 RAG 系统答错，可能错在四个完全不同的地方：语料里根本没有这个答案、答案在语料里但没被召回、召回了但在合并截断时没进最终 prompt、进了 prompt 但模型没读出来。

只测端到端答案正确率，这四种情况在数字上完全一样，但你该改的东西完全不同——前三种要动切分、向量模型、重排，最后一种才轮到提示词或换模型。方向错了，优化就是白烧钱。

>> 二、检索层指标：先看命中，再看排序

1. Hit Rate（命中率）：Top-K 里至少有一条 ground-truth 相关文档的 query 占比。最粗但最直观。
2. MRR（平均倒数排名）：第一条相关文档排在第几位，取倒数后对所有 query 求平均。排第 1 得 1 分，排第 5 得 0.2 分——关心的是「第一条对的有多靠前」。
3. NDCG@10：考虑多级相关性并对位置做对数折扣，是检索基准的统一口径（BEIR 覆盖 18 个公开数据集、9 类检索任务，全部以 nDCG@10 汇报结果）。对外汇报、跨版本对比都该用这个。
4. Recall@K：答案需要的证据有多少比例落在 Top-K 里。用来判断 K 是不是切得太小。

>> 三、生成层指标：三个必须同时看的维度

- Context Relevance（上下文相关性）：检索回来的每个片段是否与问题相关。不相关的片段会被模型顺手编织进答案，变成幻觉。
- Groundedness / Faithfulness（有据可依）：把回答拆成一条条断言（claim），逐条检查能否在检索上下文找到依据。RAGAS 的定义是「被上下文支持的断言数 / 总断言数」。
- Answer Relevance（答案相关性）：回答是否真的在回应问题，而不是答得漂亮但答非所问。

- 三者同时达标，才能说「在这个知识库的范围内，这个不产生幻觉」。注意这个结论有前提：知识库本身得是对的。

>> 四、失败归因：七类失败点

- FP1 语料缺失：问题在语料里根本没有答案，系统却硬编了一个回答。
- FP2 相关文档未被召回：答案就在库里，但排名太低没进 Top-K。
- FP3 未进入上下文：检索到了，但在合并、截断、重排环节被挤出了最终 prompt。
- FP4 未能抽取：答案就在上下文里，模型没读出来（通常是噪声太多或上下文自相矛盾）。
- FP5 格式错误：要求表格或 JSON 输出，模型忽略了指令。
- FP6 具体程度不当：答得太笼统或太细，没对上用户的实际需求。
- FP7 不完整：部分正确，漏掉了信息——虽然信息就在上下文里。

记忆锚点：FP1 到 FP3 是检索问题（文本压根没进 prompt），FP4 到 FP7 是生成与编排问题（文本进去了，管道没处理好）。这一刀切下去，优化方向就定了。

>> 五、更细的诊断：claim 级指标

- 检索侧：Claim Recall（ground-truth 答案的断言有多少被检索片段覆盖）、Context Precision（检索片段中至少支撑一条 ground-truth 断言的比例）。
- 生成侧：Faithfulness（回答断言被上下文支持的比例）、Hallucination（既不被 ground-truth 也不被检索上下文支持的断言比例）、Self-knowledge（正确但上下文里没有依据的断言比例）、Noise Sensitivity（错误断言被相关或无关片段带出来的比例）。
- Self-knowledge 是最隐蔽的信号：答案对了，但不是靠你的知识库对的。语料更新后它就会错。

>> 六、少量人工标注换统计置信度

用 LLM 合成训练数据微调轻量评判模型，给 context relevance / answer faithfulness / answer relevance 三个维度打分；再用 Prediction-Powered Inference 把几百条人工标注和大量自动评判结果结合，输出带置信区间的系统分数——而不是一个没有误差范围的裸分数。

核心要点

本期核心要点

1. 检索层与生成层必须分开测，否则分不清该改切分还是改提示词
2. 检索层：hit rate / MRR / nDCG@10 / Recall@K 四个口径
3. 生成层：context relevance / faithfulness / answer relevance 三件套同时达标
4. 七类失败点：FP1-FP3 是检索问题，FP4-FP7 是生成与编排问题
5. self-knowledge 高说明答案靠参数知识蒙对，语料更新就会错
6. 黄金集必须混入不可回答问题，否则测不出 FP1

三、测试实战

◆ 测试一：建黄金集（所有后续测试的前提）

操作：从本方知识库随机抽 100-200 个片段，用 RAGAS 测试集生成器或 LlamaIndex 的 generate_question_context_pairs 反向合成「问题 + ground-truth 片段 id」对，再人工抽检 10%-20% 剔除质量差的问题

工具：RAGAS testset generation、LlamaIndex 检索评估

指标：黄金集规模 ≥ 150 条、人工抽检不合格率 $\leq 10\%$ 、问题类型分布（事实型 / 多跳型 / 不可回答型）可控

关键：必须包含 10%-20% 的不可回答问题，否则测不出 FP1，也测不出拒答能力

◆ 测试二：检索层基线（先不看生成）

操作：用 RetrieverEvaluator 对同一黄金集跑 hit_rate / mrr / ndcg / precision / recall 五个指标，得到当前检索器的基线，再对比 BM25-only、稠密向量、混合检索、混合加重排四种配置

工具：LlamaIndex RetrieverEvaluator, metrics = [hit_rate, mrr, precision, recall, ap, ndcg]

指标：hit_rate@5 ≥ 0.85 、MRR ≥ 0.7 、nDCG@10 ≥ 0.6 作为起步线（按本方语料调整）；四种配置之间 nDCG@10 的差值，就是各环节的真实贡献

◆ 测试三：上下文增强与重排的收益验证

操作：复现 Anthropic 的 Contextual Retrieval——给每个 chunk 前置 50-100 token 的「这段在整篇文档里的位置和作用」说明，再分别建嵌入索引和 BM25 索引；然后在 top-150 候选上加 cross-encoder 重排取 top-20

工具：Anthropic Contextual Retrieval 方案、Cohere Rerank 或自建 bge-reranker

指标：以「top-20 检索失败率」为准。Anthropic 实测：只加 contextual embeddings 降 35% (5.7% 到 3.7%)，加 contextual BM25 降 49% (到 2.9%)，加重排降 67% (到 1.9%)。本方语料上复现时看能否拿到同量级降幅；拿不到，说明瓶颈不在切分而在语料覆盖或查询改写
成本侧：重排带来百毫秒级额外延迟 (top-100 候选通常是 100-300ms 量级)，要在延迟预算里先留出来

◆ 测试四：失败归因流水线（把错例分到 FP1-FP7）

操作：跑全量黄金集，对每一条错误回答按固定顺序问四个问题——（1）ground-truth

片段在库里吗？（2）在库里，那它进 Top-K 了吗？（3）进 Top-K 了，那它进最终 prompt 了吗？（4）进了 prompt，模型为什么没答对？按答案落到 FP1 到 FP7

工具：RAGChecker 做 claim 级自动诊断；或自建脚本 + LLM 归因提示词

指标：七类失败的分布比例。经验上若 FP1 + FP2 合计超过 40%，优化重点应在语料覆盖与切分检索，而不是提示词；若 FP4 占比高，才轮到提示词与上下文压缩

输出物：一张「失败点 × 数量 × 典型样例」的归因表，比一个笼统的 82% 正确率有用得多

◆ 测试五：噪声敏感度与抗干扰测试

操作：构造三组输入——只给相关片段、相关片段加 2 条无关片段、相关片段加 5 条矛盾片段（把某条制度的关键数字改错），看回答如何变化

工具：RAGChecker 的 noise_sensitivity_in_relevant / noise_sensitivity_in_irrelevant，或 RAGAS 的 Noise Sensitivity

指标：噪声敏感度（错误断言被无关片段带出的比例）应低于 10%；矛盾片段注入后正确回答率下降幅度低于 15%，且模型应指出矛盾而不是随机选一条照抄

◆ 测试六：不可回答问题与拒答率

操作：用黄金集里的不可回答子集（语料里没有答案但话题相近的问题）单独跑一轮，统计正确拒答、强行作答、答非所问三类比例

工具：黄金集不可回答子集 + 人工与 LLM 双重判定

指标：正确拒答率 $\geq$ 85%、强行作答率（编造答案） $\leq$ 10%。这类问题是 FP1 的直接探针——很多系统在能回答的问题上表现很好，一遇到语料外的问题就编

◆ 测试七：改动前后回归（唯一能证明「改好了」的测试）

操作：任何一次改动（换 embedding、改 chunk size、加 reranker、改提示词）都在同一份黄金集上跑检索层与生成层全指标，与基线逐项对比

工具：RAGAS 评估管线，每次结果版本化保存

指标：nDCG@10 与 faithfulness 不得同时下降；任一指标下降超过 3 个百分点即视为回归，定位原因后再合入

提醒：黄金集要定期补充线上真实失败 case，否则指标会虚高

四、常见坑点

坑点一：用生成端指标诊断检索问题

faithfulness 掉了，很多人的第一反应是改提示词。但如果 ground-truth 片段压根没被召回，提示词怎么改都救不回来——模型只能用参数知识编。而且已有研究在长文档、学术论文语料上发现 RAGAS 的 faithfulness 在特定配置下可靠性有限，把它当唯一裁判有风险。正确顺序：先跑检索层指标（hit_rate / MRR / nDCG@10），确认检索没问题，再看生成层。

坑点二：答案对了就以为检索对了

RAGChecker 专门有一个 self-knowledge 指标，衡量「正确但检索上下文里没有依据」的断言比例。这个数高，说明模型在靠预训练知识答题——在你自己的语料上看着对，一旦制度更新、口径变化，它就会给出过时但自信的答案。银行场景里这是最危险的一类假成功。正确率必须

和 faithfulness、self-knowledge 一起看。

坑点三：不做对照就上语义分块

「语义分块一定比固定长度好」是个流传很广的误解。有系统评测的结论是：语义分块的计算开销并不能被稳定的效果提升所证明，在更接近真实文档的非合成数据集上，固定长度分块常常更好。而 chunk 大小的最优值又高度依赖任务类型——有研究发现 64-128 token 的小分块适合简短事实型问答，512-1024 token 的大分块适合需要整体上下文理解的场景。做法：把 chunk size (256 / 512 / 1024) × overlap 做成网格，在同一份黄金集上跑 nDCG@10 对照，用数据选，不要用直觉选。

坑点四：黄金集里全是「能回答的问题」

只挑能回答的问题建测试集，等于把

FP1（语料缺失）这一整类风险排除在测试范围之外。上线后用户第一个问的，往往就是语料里没有的问题。黄金集必须按比例混入不可回答问题，并且单独统计拒答率。

坑点五：LLM 评判的随机性没被控制

RAGAS、ARES 这类指标都是 LLM 打分，换一个 judge 模型版本、温度没归零、prompt 措辞变一下，分数能漂几个百分点，看起来像优化了，其实是噪声。做法：固定 judge 模型与版本、温度设为 0、同一配置跑 3 次取均值、在结果里记录 judge 配置；关键决策点用 ARES 的 PPI 思路补几百条人工标注，给分数配上置信区间，而不是拿一个裸数字下结论。

五、延伸阅读

- RAGAS 论文 (reference-free RAG 评估) : <https://arxiv.org/abs/2309.15217>
- RAGChecker 论文 (claim 级诊断指标) : <https://arxiv.org/abs/2408.08067>
- RAGChecker 代码 (Amazon Science) : <https://github.com/amazon-science/RAGChecker>
- ARES 论文 (PPI 统计置信区间) : <https://arxiv.org/abs/2311.09476>
- 七类 RAG 失败点 (Barnett 等) : <https://arxiv.org/abs/2401.05856>
- BEIR 基准论文 (统一 nDCG@10) : <https://arxiv.org/abs/2104.08663>
- Anthropic Contextual Retrieval: <https://www.anthropic.com/engineering/contextual-retrieval>
- TruLens RAG Triad: https://www.trulens.org/getting_started/core_concepts/rag_triad/
- LlamaIndex 检索评估指南: https://developers.llamaindex.ai/python/examples/evaluation/retrieval/retriever_eval/
- 语义分块是否值得其计算成本: <https://arxiv.org/abs/2410.13070>
- chunk size 多数据集分析: <https://arxiv.org/abs/2505.21700>