第 12 课：RAG 总体流程

1. RAG 是什么？

RAG 全称是 **Retrieval-Augmented Generation**，检索增强生成。

它的核心思想是：先从外部知识库检索相关资料，再把检索到的资料作为上下文交给大模型生成答案。

普通 LLM 回答：用户问题 → LLM → 答案

RAG 回答：

用户问题
→ 检索知识库
→ 找到相关文档片段
→ 把片段塞进 Prompt
→ LLM 基于证据生成答案

面试说法：RAG 是检索增强生成。它不是只依赖模型参数记忆，而是先从企业知识库、文档库或数据库中检索相关内容，再让大模型基于检索结果回答。这样可以缓解知识过时、企业私有知识缺失和幻觉问题。

2. 为什么需要 RAG？

因为大模型有几个问题：

1. 不知道企业内部私有知识
2. 训练知识可能过时
3. 容易编造答案
4. 无法天然给出引用来源
5. 每次为了新知识微调成本太高

RAG 的作用：

1. 接入企业知识库
2. 更新文档比重新训练模型更轻
3. 让回答基于证据
4. 可以返回引用来源
5. 降低幻觉

面试说法：企业场景下很多知识不在模型训练数据里，比如制度文档、报销规则、IT 故障手册。RAG 可以把这些外部知识接入模型，知识更新时只需要更新索引，不一定重新训练模型。同时通过引用和拒答机制，可以降低幻觉风险。

3. RAG 和微调有什么区别？

RAG 解决的是：

知识接入、事实查询、文档问答、可追溯引用。

微调解决的是：

模型行为风格、输出格式、领域表达习惯、特定任务能力。

对比：

维度	RAG	微调
核心目的	接入外部知识	改变模型行为
知识更新	更新文档和索引	重新训练或增量训练
适合场景	企业文档问答、政策查询	固定格式输出、分类、风格对齐
可追溯性	可以引用来源	不天然可追溯
成本	相对低	相对高
幻觉控制	靠检索、引用、拒答	仍可能幻觉

面试说法：RAG 更适合解决知识问题，尤其是企业私有文档、政策制度、知识库问答；微调更适合解决行为问题，比如让模型稳定按照某种格式输出、学习特定任务风格或专业表达。知识频繁更新时优先考虑 RAG，

而不是每次都微调。

4. RAG 的基本流程

完整流程通常分两部分：**离线入库** 和 **在线问答**。

离线入库

原始文档
→ 文档解析
→ 清洗
→ chunk 切分
→ embedding 向量化
→ 写入向量库

例如你的项目：

PDF/制度文档
→ 抽取文本
→ structured_hybrid chunk
→ SentenceTransformer 编码
→ 写入 Qdrant
→ payload 保存 doc_id/page/section_path/text

在线问答

用户问题
→ query embedding
→ 向量检索 / BM25 检索
→ 混合检索
→ rerank
→ 取 Top-K 证据
→ 组织 Prompt
→ LLM 生成答案
→ 返回答案 + citation

面试说法：RAG 一般分为离线入库和在线问答。离线阶段把文档解析、清洗、切 chunk、向量化并写入向量库；在线阶段对用户问题做检索，召回相关 chunk，再经过 rerank 筛选证据，把证据放入 Prompt 让大模型回答，最后返回答案和引用。

5. Chunk 是什么？

Chunk 是文档切分后的文本片段。

为什么要切 chunk？

因为：

1. 文档太长，不能整个塞进 Prompt
2. 向量检索通常以片段为单位
3. 太长会稀释语义
4. 太短会丢上下文

面试说法：chunk 是文档切分后的文本片段。RAG 通常不是把整篇文档直接向量化，而是切成合适大小的片段。chunk 太长会导致语义不聚焦，太短又会丢失上下文，所以需要根据文档结构、段落、标题和 token 长度设计切分策略。

6. Embedding 是什么？

Embedding 是把文本变成向量。

目的：让语义相近的文本，在向量空间里距离更近。

例如：

“怎么报销差旅费”
“差旅费报销规则是什么”

它们字面不完全一样，但语义接近，embedding 后向量距离应该比较近。

面试说法：embedding 是把文本映射成稠密向量，用于语义相似度检索。RAG 中会把文档 chunk 和用户问

题都编码成向量，再通过向量相似度找到语义相关的内容。

7. 向量库是什么？

向量库用于存储和检索 embedding。

常见向量库：

- Qdrant
- Milvus
- FAISS
- Chroma
- Pinecone
- Weaviate

你的项目用的是：

- Qdrant

向量库通常存：

- 向量
- 文本
- doc_id
- page
- section_path
- metadata
- 权限字段

面试说法：向量库用于存储文档 chunk 的 embedding，并支持相似度检索。除了向量本身，还会保存文本内容和 metadata，比如文档 ID、页码、章节、权限标签等，方便后续引用和权限过滤。

8. Top-K 是什么？

Top-K 指检索时返回最相关的 K 个结果。

例如：

- Top-5：返回最相关的 5 个 chunk
- Top-20：返回最相关的 20 个 chunk

Top-K 太小：

- 可能召回不到答案

Top-K 太大：

- 噪声变多
- Prompt 变长
- 成本升高

面试说法：Top-K 是检索返回的候选数量。K 太小容易漏召回，K 太大又会引入噪声、增加重排和生成成本。实际项目中通常会先召回较多候选，再用 rerank 选出更少的高质量证据。

9. RAG 为什么会答错？

常见原因：

1. 文档解析错误
2. chunk 切分不合理
3. embedding 模型不适合
4. 检索没有召回正确片段
5. rerank 排序错误
6. Prompt 证据组织不好
7. LLM 没有严格基于证据回答
8. 没有拒答机制
9. 权限过滤错误

面试说法：RAG 答错不一定是大模型生成错了，也可能是前面的检索链路出了问题。需要分层排查：文档是否解析正确，chunk 是否合理，检索是否召回正确证据，rerank 是否把正确证据排前面，Prompt 是否明确要求基于证据回答，以及是否有拒答兜底。

第 13 课：文档解析与 chunk 切分

文档解析和 chunk 切分质量直接决定 RAG 上限

1. 文档解析是什么？

文档解析就是把原始文件转成可检索的文本和结构化信息。

原始文件可能是：

- PDF
- Word
- Excel
- HTML
- Markdown
- 扫描件图片
- 网页
- 数据库记录

解析结果通常包括：

- 正文文本
- 标题层级
- 段落
- 表格
- 页码
- 图片说明
- 文档元信息

面试说法：文档解析是 RAG 入库的第一步，目的是把 PDF、Word、网页等原始资料转成干净、可切分、可检索的文本，同时尽量保留标题、页码、章节、表格等结构信息，方便后续 chunk、引用和权限过滤。

2. PDF 为什么难解析？

PDF 本质上更像“版式文件”，不是天然的结构化文档。

常见问题：

- 段落顺序错乱
- 页眉页脚污染
- 换行断裂
- 表格被拆散
- 双栏顺序混乱
- 图片里的文字读不出来
- 扫描件需要 OCR
- 页码、标题层级丢失

面试说法：PDF 解析难点在于它更偏版式展示，而不是结构化文本。解析时容易出现段落顺序错乱、页眉页脚污染、表格拆散、双栏混乱、扫描件需要 OCR 等问题。这些问题会直接影响后续 chunk 和检索质量。

3. 为什么要清洗文本？

解析出来的文本通常有噪声：

- 页眉页脚
- 重复目录
- 多余空格
- 异常换行
- 乱码
- 广告
- 版权声明
- 无意义符号

如果不清洗：

embedding 被噪声污染
chunk 内容不完整
检索命中无关内容
LLM 生成答案混乱

面试说法：文档清洗是为了去掉页眉页脚、重复目录、多余空格、异常换行、乱码等噪声，避免这些内容进入 embedding 和 Prompt，影响检索和生成质量。

4. chunk 切分的目标是什么？

chunk 切分不是随便按长度切。

目标是：

语义完整
长度适中
便于检索
便于引用
保留上下文
减少噪声

面试说法：chunk 切分的目标是在语义完整和长度可控之间做平衡。好的 chunk 应该尽量包含完整的规则或语义单元，同时不要太长，以免语义稀释；也不要太短，以免上下文断裂。

5. 常见 chunk 切分方法

1）固定长度切分

优点：

简单
稳定
容易实现

缺点：

可能切断句子、段落和规则
不理解文档结构

2）滑动窗口 overlap

优点：

缓解边界信息丢失
提高召回概率

缺点：

增加索引体积
引入重复内容
可能增加检索噪声

面试说法：overlap 可以缓解切块边界导致的信息丢失，但会增加向量数量和存储成本，也可能让检索结果出现重复内容。

3）按段落 / 标题切分

优点：

语义更完整
适合制度文档、政策文档、FAQ

缺点：

需要解析结构
不同文档格式差异大

4）递归切分 Recursive Splitter

常见逻辑：

先按标题切
再按段落切
再按句子切

最后按长度兜底

面试说法：递归切分会优先按高层语义结构切分，比如标题和段落；如果某段仍然太长，再按句子或长度继续切。它比纯固定长度更容易保留语义完整性。

6. chunk_size 怎么选？

chunk_size 没有固定答案，要看：

文档类型
embedding 模型最大长度
LLM 上下文窗口
问题粒度
是否有 rerank
是否需要引用精度

经验：

FAQ / 短规则：200 ~ 500 tokens
制度文档：400 ~ 800 tokens
长报告：800 ~ 1200 tokens

但重点不是死记数字，而是理解取舍。

面试说法：chunk_size 要根据文档类型、问题粒度、embedding 模型长度和引用需求调参。太小容易上下文不足，太大容易语义稀释。制度类文档通常会选择几百 tokens 左右，并结合标题和段落结构切分，而不是机械按长度切。

7. overlap 怎么选？

overlap 是相邻 chunk 的重叠部分。

作用：

缓解边界信息丢失
保留上下文连续性

但不能过大：

索引体积变大
重复召回变多
成本增加

面试说法：overlap 用于缓解边界切断问题，但不是越大越好。overlap 太大会增加索引体积和重复召回。通常会根据 chunk_size 设置一个较小比例，比如 10% ~ 20%，并通过评测观察召回效果。

8. metadata 为什么重要？

metadata 是 chunk 附带的元信息。

常见字段：

doc_id
title
source_file
page
page_start
page_end
section_path
category
version
status
permission_scope
created_at

作用：

引用 citation
权限过滤

版本过滤
审计追踪
结果展示
问题排查

面试说法：RAG 入库不能只存文本和向量，还要保存 metadata，比如 doc_id、页码、章节路径、版本、权限范围等。metadata 是 citation、权限过滤、版本控制和审计追踪的基础。

9. 表格怎么处理？

表格是 RAG 难点。

如果直接按普通文本解析，可能变成：

姓名 金额 日期
张三 100 2024
李四 200 2025

还好。

但复杂表格可能会丢失行列关系。

常见处理：

转 Markdown 表格
转 HTML 表格
转自然语言描述
按行切分
为表格生成摘要
保留表格页码和标题

面试说法：表格不能简单按普通文本处理，因为行列关系容易丢失。常见做法是把表格转成 Markdown 或 HTML，或者转成自然语言描述；对于复杂表格，可以保留表格标题、页码和行列结构，必要时单独作为 chunk 入库。

10. 扫描件和图片文字怎么办？

扫描件没有可直接抽取的文本，需要 OCR。

但 OCR 有问题：

识别错误
漏字
表格错乱
版面顺序错
成本高
速度慢

面试说法：对扫描件或图片中的文字，需要先做 OCR，再进入清洗和切分流程。但 OCR 结果可能有错字、漏字、版面顺序混乱，所以最好结合人工抽检或质量评估，避免低质量 OCR 文本污染知识库。

11. chunk 质量怎么评估？

可以看：

检索召回率
答案命中率
引用准确率
人工抽检
chunk 是否语义完整
是否包含噪声
是否过短/过长

面试说法：chunk 质量最终要通过 RAG 效果评估。可以构建测试问题，看正确证据是否能被召回，答案是否基于正确 chunk，citation 是否准确。同时也要人工抽检 chunk，看是否存在断句、噪声、表格错乱、标题丢失等问题。

第 14 课：Embedding 与向量库

1. Embedding 是什么？

Embedding 是把文本转换成向量。

例如：

“如何报销差旅费”

“差旅费报销流程是什么”

字面不同，但语义接近。Embedding 的目标就是让它们在向量空间里距离更近。

面试说法：Embedding 是把文本映射成稠密向量，用于表示文本语义。RAG 中会把文档 chunk 和用户 query 都编码成向量，再通过向量相似度找到语义相关的文档片段。

2. 为什么不能只靠关键词匹配？

关键词匹配看字面重合。

例如用户问：

“电脑坏了怎么处理？”

文档写的是：

“终端设备故障需提交 IT 工单。”

关键词不完全一样，但语义相关。Embedding 可以帮助召回这种表达不同但语义相近的内容。

面试说法：关键词检索适合精确匹配术语、编号、人名、代码，但对同义表达和语义相近的问题不够友好。Embedding 检索可以补足这一点，通过语义相似度召回字面不同但含义接近的内容。

3. 文档向量和 query 向量为什么要用同一个模型？

因为向量空间必须一致。

如果文档 chunk 用模型 A 编码，用户问题用模型 B 编码，它们的向量不在同一个语义空间里，相似度就没有可比性。

面试说法：文档向量和 query 向量通常要用同一个 embedding 模型生成，保证它们处在同一个向量空间。否则相似度计算没有意义，检索结果会不稳定。

4. 相似度怎么算？

常见有：

cosine similarity：余弦相似度

dot product：点积

euclidean distance：欧氏距离

RAG 里常见的是余弦相似度或点积。

面试说法：

向量检索本质是计算 query 向量和 chunk 向量的相似度，常见方式有余弦相似度、点积和欧氏距离。相似度越高，说明语义越接近。

5. 向量维度是什么？

Embedding 模型输出的向量长度就是维度。

例如：

384 维

768 维

1024 维

1536 维

维度不是越高越好。

高维可能表达能力更强，但也会带来：

存储更大

检索更慢

成本更高

面试说法：向量维度由 embedding 模型决定。维度越高不一定越好，需要在语义效果、存储成本和检索性能之间权衡。

6. 向量库是什么？

向量库用于存储向量，并支持相似度检索。
常见向量库：

- Qdrant
- Milvus
- FAISS
- Chroma
- Pinecone
- Weaviate

向量库一般存：

- id
- vector
- payload / metadata
- 原始 chunk 文本
- doc_id
- page
- section_path
- permission_scope

面试说法：向量库负责存储 chunk embedding，并提供高效的相似度检索能力。实际项目中不仅存向量，还要存 payload，比如原文、文档 ID、页码、章节路径、权限字段等，方便引用、过滤和审计。

7. Qdrant 是什么？

Qdrant 是一个向量数据库，支持向量检索和 payload 过滤。
你的项目可以这样说：我的项目使用 Qdrant 作为向量库。入库时把每个 chunk 的 embedding 写入 Qdrant，同时在 payload 中保存 text、doc_id、page、section_path、category、version、permission_scope 等字段。在线检索时先做向量相似度搜索，同时可以根据 metadata 做过滤。

8. payload 过滤是什么？

payload 过滤就是检索时先按 metadata 限制范围。
例如：

- 只检索 status=active 的文档
- 只检索 category=finance 的文档
- 只检索当前用户有权限访问的文档
- 只检索某个 version 的文档

面试说法：payload 过滤是企业 RAG 很重要的能力。检索前可以根据文档状态、类别、版本和用户权限过滤候选范围，避免召回过期文档或用户无权限访问的内容。

层级	字段类型	包含内容	作用
第一层 强索引层	标量字段 (Scalar Index)	权限 (Tenant ID)、文档 ID、时间戳	用于预过滤。必须建立倒排索引，执行速度极快。
第二层 向量索引层	Vector	Float 数组	用于 ANN 相似度计算。
第三层 存储负载层	Payload / Metadata	章节路径、页码、文本摘要、作者名、标签	只用于展示或后过滤（Post-filter）。不参与检索前的粗筛，只作为结果返回给用户做展示。

9. 为什么要重建索引？

当以下内容变化时，可能需要重建或增量更新索引：

- 新增文档
- 删除文档
- 文档内容更新

chunk 策略变化
embedding 模型变化
metadata 字段变化

尤其注意：如果换了 embedding 模型，旧向量和新向量不在同一个空间，通常需要重新向量化并重建索引。

面试说法：文档内容更新、chunk 策略变化、embedding 模型变化时，都可能需要更新索引。尤其是更换 embedding 模型后，旧向量和新向量不在同一语义空间，通常要重新对文档向量化并重建索引。

丢弃旧的索引结构（如 HNSW 图、IVF 倒排列表），利用已有的数据重新构建一张新的查找表

10. 向量检索有什么局限？

向量检索擅长语义相似，但不擅长所有问题。

局限包括：

精确编号、代码、日期、人名可能不如关键词检索
企业黑话如果 embedding 没学过，可能召回差
长 query 可能语义混杂
相似但不相关的片段可能被召回
权限过滤如果缺失会造成数据泄露

面试说法：向量检索适合语义召回，但对编号、代码、精确术语、人名、日期等字面匹配问题不一定稳定。所以企业 RAG 常结合 BM25 关键词检索，再用 RRF 或 rerank 做融合排序。

第 15 课：BM25 / Dense / Hybrid Search / RRF

1. Dense 检索是什么？

Dense 检索就是向量检索。

流程：

用户问题
→ embedding 模型编码成 query vector
→ 向量库中查找相似 chunk vector
→ 返回语义相近的 chunk

优点：

能处理同义表达
能召回语义相近但字面不同的内容
适合自然语言问题

缺点：

对错误码、编号、日期、人名、专有名词不一定稳定
可能召回“看起来语义相似但事实不相关”的内容
依赖 embedding 模型质量

面试说法：Dense 检索是基于 embedding 的语义检索，适合召回表达不同但语义相近的内容。但它对编号、错误码、专有名词等精确匹配问题不一定稳定。

2. BM25 是什么？

BM25 是一种经典关键词检索算法，可以理解为 TF-IDF 的改进版。

它主要看：

query 词是否出现在文档中
出现频率
词本身是否稀有
文档长度归一化

比如用户问：

VPN-403 错误怎么处理？

BM25 很擅长找包含：

VPN-403
403
VPN

错误

这些关键词的文档。

面试说法：BM25 是一种基于关键词匹配的稀疏检索方法，适合处理错误码、编号、专有名词、人名、日期等精确匹配问题。它不依赖 embedding 模型，但对同义表达和语义泛化能力较弱。

3. Sparse vs Dense

类型	代表	擅长	不擅长
Sparse 检索	BM25	关键词、编号、错误码、精确术语	同义表达、语义泛化
Dense 检索	向量检索	语义相似、自然语言问法	精确编号、专有词、代码

面试说法：Sparse 检索擅长字面匹配，Dense 检索擅长语义匹配。企业知识库里既有自然语言问题，也有错误码、制度编号、工单编号，所以通常会把两者结合起来。

4. 为什么要混合检索 Hybrid Search？

因为单一路线都有短板。

用户问：

差旅费怎么报销？

Dense 检索可能很好。

用户问：

VPN-403 怎么处理？

BM25 可能更好。

所以混合检索：

Dense 检索召回一批

BM25 检索召回一批

融合排序

得到更稳的候选结果

面试说法：混合检索是把 Dense 语义检索和 BM25 关键词检索结合起来。Dense 负责召回语义相似内容，BM25 负责补充精确匹配能力。这样可以同时覆盖自然语言问法和企业专有词、错误码、编号等场景。

5. RRF 是什么？

RRF 全称是 Reciprocal Rank Fusion，倒数排名融合。

它不直接关心原始分数，而是关心排名。

公式可以简单理解为：

score = 1 / (k + rank)

rank 越靠前，分数越高。

如果一个 chunk 在 Dense 检索和 BM25 检索里都排得靠前，那融合后排名通常会更高。

面试说法：RRF 是一种排名融合方法，用来融合多个检索器的结果。它不直接比较不同检索器的原始分数，而是根据各自排名计算融合分数。这样可以避免 Dense 分数和 BM25 分数尺度不同导致难以直接相加的问题。

6. 为什么不能直接把 Dense 分数和 BM25 分数相加？

因为两个分数不在同一个尺度。

比如：

Dense cosine score: 0.78

BM25 score: 15.3

直接相加没有意义。

RRF 只看排名，不看原始分数，所以更稳。

面试说法：Dense 和 BM25 的分数来源不同、尺度不同，不能简单相加。RRF 用排名而不是原始分数做融合，因此更适合融合多路检索结果

7. Hybrid Search 的一般流程

用户问题

→ Dense 检索 Top-N

→ BM25 检索 Top-N
→ 合并候选
→ RRF 计算融合分数
→ 得到融合后的候选列表
→ rerank
→ 取 Top-K 证据
→ 生成答案

面试说法：
我的项目里会先分别做 Dense 检索和 BM25 检索，各自召回一批候选 chunk，然后用 RRF 根据排名融合结果，再交给 CrossEncoder rerank 重新排序，最后取 Top-K 证据进入 Prompt。

8. Recall 和 Precision

两个常见指标：

Recall：正确证据有没有被召回
Precision：召回结果里有多少是相关的

RAG 里召回阶段更重视 Recall。
因为：

正确证据如果没被召回，后面的 rerank 和 LLM 都救不了。

但 Recall 太高也不行，因为会带来太多噪声。
面试说法：检索阶段首先要保证召回率，也就是正确证据能进候选集；随后通过 rerank 和 Top-K 控制精度，减少噪声进入 Prompt。

9. Top-N 和 Top-K 的区别

常见说法：

Top-N：检索阶段召回的候选数量
Top-K：最终送入 Prompt 的证据数量

比如：

Dense Top-20
BM25 Top-20
RRF 融合后候选 30 个
rerank 后取 Top-5 进入 Prompt

面试说法：检索阶段通常会取较大的 Top-N 保证召回，然后 rerank 后取较小的 Top-K 放入 Prompt。这样既能提高正确证据被召回的概率，又能控制 Prompt 噪声和 token 成本。

10. 混合检索的代价

混合检索更稳，但也有代价：

实现复杂度更高
需要维护 BM25 索引和向量索引
延迟增加
候选去重和融合更复杂
需要评测调参

面试说法：混合检索能提升召回稳定性，但会增加系统复杂度和延迟，需要维护关键词索引和向量索引，也要通过评测调节各路召回数量、RRF 参数和 rerank 阈值

第 16 课：Rerank / CrossEncoder 重排序

1. 为什么召回后还要 rerank？

因为第一阶段检索主要目标是 **召回**，也就是先把可能相关的 chunk 找出来。
但召回结果里会有：

语义相似但不是真正答案的 chunk
关键词匹配但上下文不相关的 chunk
重复 chunk
排序不够准确的 chunk

所以需要 rerank 重新排序，把最相关的证据排到前面。
面试说法：检索阶段通常先用 Dense 和 BM25 召回较多候选，保证正确证据尽量进入候选集。但召回结果可能有噪声，所以需要 rerank 对 query 和候选 chunk 做更精细的相关性判断，把真正能回答问题的证据排到前面。

2. Bi-Encoder 和 CrossEncoder 区别

Bi-Encoder

Dense 检索通常是 Bi-Encoder。
流程：

- query 单独编码成向量
- chunk 单独编码成向量
- 再计算向量相似度

优点：

- 快
- 可以提前把文档向量入库
- 适合大规模召回

缺点：

- query 和 chunk 没有一起细粒度交互
- 相关性判断不够精细

CrossEncoder

CrossEncoder 是把 query 和 chunk 拼在一起输入模型。
流程：

- [query, chunk] → CrossEncoder → relevance score

优点：

- 相关性判断更准确
- 能看 query 和 chunk 的细节匹配
- 适合重排序

缺点：

- 慢
- 不能提前为所有 chunk 预计算
- 只能对少量候选做 rerank

面试说法：Bi-Encoder 是把 query 和文档分别编码成向量，适合大规模快速召回；CrossEncoder 是把 query 和候选 chunk 拼在一起输入模型，直接判断相关性，精度更高但速度更慢。所以一般先用 Bi-Encoder 召回 Top-N，再用 CrossEncoder 对少量候选 rerank。

3. rerank 放在哪一层？

典型流程：

- 用户问题
- → Dense Top-N
- → BM25 Top-N
- → RRF 融合
- → CrossEncoder rerank
- → 取 Top-K
- → 构建 Prompt
- → LLM 生成

注意：rerank 不是替代检索，而是对检索候选做二次排序。如果一开始没有召回正确证据，rerank 也救不了。

4. rerank 的代价

rerank 会提高质量，但也有代价：

- 延迟增加
- 模型推理成本增加
- 候选数量越多越慢

需要额外部署或加载 rerank 模型

所以不能把全量文档都送去 rerank。

一般做法是：

先召回 Top-20 / Top-50

再 rerank

最后取 Top-3 / Top-5 进 Prompt

面试说法：rerank 精度更高但成本更高，所以通常只对召回阶段得到的小规模候选做重排序，而不是对全量文档做 rerank。实际项目中要根据延迟和效果评测调整候选数量。

5. rerank 怎么评估有没有用？

看 rerank 前后：

正确证据排名是否提升

Top-K 证据质量是否提升

答案正确率是否提升

引用准确率是否提升

延迟是否可接受

常见指标：

Recall@K

MRR

NDCG

Answer Accuracy

Citation Accuracy

Latency

你面试不用全背，重点说：看正确证据是否更靠前，以及最终答案和引用是否更准确。

6. 你的项目怎么讲？

你可以这样说：我的项目采用 Dense + BM25 + RRF + CrossEncoder rerank。Dense 和 BM25 负责召回候选，RRF 负责融合两路排序结果，CrossEncoder rerank 负责对候选 chunk 做更精细的相关性判断。因为 CrossEncoder 会把用户问题和 chunk 一起输入模型，所以比单纯向量相似度更准确，但速度更慢，因此我只对融合后的 Top-N 候选做 rerank，再取 Top-K 证据进入 Prompt。

如果 rerank 后延迟变高：

减少进入 rerank 的候选数量

换更轻量的 rerank 模型

做批量推理

缓存高频 query 的 rerank 结果

只在复杂问题、低置信度问题上启用 rerank

第 17 课：Prompt 证据组织 / Citation / 拒答

1. 为什么证据组织很重要？

检索到正确 chunk 不代表 LLM 一定能答好。

如果 Prompt 里：

证据太多

顺序混乱

无关 chunk 很多

没有编号

没有明确要求基于证据回答

没有引用格式约束

模型仍然可能乱答。

面试说法：RAG 不是把检索结果简单拼进 Prompt 就结束了。证据组织会影响模型是否能正确使用上下文。通常需要控制证据数量、排序、编号、来源信息，并明确要求模型只基于证据回答。

2. Prompt 里证据怎么组织？

常见格式：

```
你是企业制度问答助手。
请只基于以下证据回答用户问题。
如果证据不足，请回答“知识库中没有找到依据”。

用户问题：
{question}

证据：
[1] 来源：财务制度.pdf，第 3 页，报销标准
内容：……

[2] 来源：差旅制度.pdf，第 5 页，交通费
内容：……

回答要求：
1. 只基于证据回答
2. 不要编造证据外内容
3. 每个关键结论后标注引用，如 [1]
4. 如果证据不足，拒答
```

核心是：

```
证据编号
来源信息
正文内容
回答约束
引用格式
拒答规则
```

3. citation 是什么？

citation 就是引用来源。

RAG 里常见引用：

```
根据《差旅费管理办法》第 3 页，员工出差住宿费需要在标准内报销。[1]
```

citation 依赖前面入库保存的 metadata：

```
doc_id
title
source_file
page
section_path
chunk_id
```

面试说法：citation 是 RAG 答案的引用来源，用来告诉用户答案依据来自哪份文档、哪一页、哪一节。它依赖入库时保存的 metadata，比如 doc_id、page、section_path 和 source_file。

4. citation 有什么价值？

```
1. 可追溯
2. 方便用户核查
3. 降低幻觉风险
4. 方便审计
5. 便于发现错误证据
```

面试说法：citation 能让答案可追溯，用户可以核查来源；出错时也能判断是检索错了、证据错了，还是模型生成错了。企业场景里这比普通聊天更重要。

5. 为什么要拒答？

如果知识库没有证据，模型不能硬编。

拒答场景：

- 没有检索到相关 chunk
- 相关性分数太低
- 检索到了但证据不能回答问题
- 用户问超出知识库范围
- 用户无权限查看相关证据

面试说法：RAG 系统必须有拒答机制。证据不足时，与其让模型编造，不如明确说明知识库中没有找到依据。企业知识库问答更看重可靠性，而不是每个问题都强行回答。

6. 拒答怎么实现？

常见做法：

1. 检索阶段设置相似度阈值
2. rerank 分数低于阈值拒答
3. Top-K 证据为空拒答
4. Prompt 明确证据不足时拒答
5. 生成后检查答案是否带有效引用

注意：不要只靠 Prompt 拒答，最好检索层和生成层都做控制。

7. 引用不等于真实可靠

模型可能会出现：

- 答案对，但引用错
- 引用存在，但证据不支持结论
- 编造不存在的引用编号
- 多个引用混乱

所以需要：

- 引用编号只能来自提供的证据
- 答案关键结论必须有引用
- 评测 citation accuracy
- 必要时做后处理校验

面试说法：citation 不是简单让模型写个编号就完了，还要确保引用编号来自实际检索证据，并且被引用证据能支持对应结论。否则会出现“看起来有引用，但引用不支持答案”的问题。

8. 你的项目怎么讲？

你可以这样说：我的项目在生成阶段不是简单把 chunk 拼给模型，而是把 rerank 后的 Top-K 证据带上 doc_id、page、section_path 和内容一起组织进 Prompt。Prompt 明确要求模型只基于证据回答，并在关键结论后给出引用。如果检索证据不足或相关性低，就触发拒答兜底，避免模型编造。同时系统会记录 kb_query 和 trace_id，方便后续排查是检索、rerank 还是生成阶段出了问题。

第 18 课：RAG 评测

1. 为什么 RAG 需要评测？

RAG 链路很长：

文档解析 → chunk → embedding → 检索 → rerank → Prompt → 生成 → 引用 → 拒答

任何一层出问题都会导致答案错。

面试说法：RAG 不能只靠主观试几个问题判断效果，需要构建评测集，从检索、生成、引用、拒答和延迟多个维度量化评估。

2. RAG 评测分几层？

核心分三层：

1. 检索评测：正确证据有没有被召回
2. 生成评测：答案是否正确、是否忠实于证据
3. 工程评测：延迟、成本、稳定性

企业 RAG 还要看：

4. citation 是否准确
5. 权限过滤是否正确
6. 证据不足时是否合理拒答

3. 检索指标

最重要：

Recall@K：正确证据是否出现在前 K 个结果中
MRR：正确证据排得是否靠前
NDCG：排序整体质量

面试可以简单说：检索阶段我最关心 Recall@K，因为如果正确证据没有被召回，后面的 rerank 和 LLM 很难补救。

4. 生成指标

常见看：

Answer Accuracy：答案是否正确
Faithfulness：答案是否忠实于证据
Citation Accuracy：引用是否支持答案
Refusal Accuracy：该拒答时是否拒答，不该拒答时是否回答

注意：

答案对但引用错，在企业 RAG 里也不能算完全通过。

5. 怎么构建评测集？

不要只随便问几个问题。

评测集应该覆盖：

普通事实问答
多条款综合问题
数字/日期/金额问题
专有名词/错误码问题
证据不足问题
权限过滤问题
版本冲突问题
相似问题改写

每条最好包含：

question
gold_answer
gold_doc_id / gold_chunk_id
expected_citation
should_refuse
category

6. 你的项目怎么讲？

你可以这样说：我的项目会构建一批 RAG 评测问题，每条问题标注标准答案、应该命中的文档或 chunk、是否应该拒答。评测时先看检索层的 Recall@K、MRR，判断正确证据是否被召回并排到前面；再看生成层的答案正确率、faithfulness、citation accuracy 和拒答准确率；最后看延迟和成本。这样可以定位问题是出在 chunk、检索、rerank，还是 Prompt 生成阶段。

第 19 课：企业权限过滤与工程优化

1. 为什么企业 RAG 必须做权限过滤？

企业文档不是所有人都能看。

例如：

普通员工：只能看通用制度、自己的工单
财务人员：能看财务制度
管理员：能看全部文档和审计日志

如果没有权限过滤，用户可能通过自然语言问出无权限内容：

“告诉我财务部的薪酬制度”

“查一下别人的工单详情”

面试说法：企业 RAG 不能只考虑检索准不准，还要考虑用户有没有权限看这些证据。检索前或检索时必须结合 current_user 做权限过滤，避免无权限文档进入候选集和 Prompt。

2. 权限过滤放在哪里？

最好尽量靠前。

常见位置：

1. 检索前：根据用户角色、部门、租户、权限范围构造 filter
2. 向量库检索时：用 payload filter 过滤 permission_scope
3. 生成前：再次检查 Top-K 证据是否都属于用户可访问范围
4. 审计层：记录用户检索了哪些文档

面试说法：权限过滤最好在检索阶段就做，而不是等模型生成后再拦截。我的做法是把文档权限信息写入 metadata，检索时根据 current_user 构造 payload filter，只召回用户有权限访问的 chunk，进入 Prompt 前再做一次校验。

3. 多租户是什么

多租户就是一个系统服务多个企业或组织。

例如：

tenant_a 的用户只能检索 tenant_a 的文档

tenant_b 的用户只能检索 tenant_b 的文档

RAG 中必须保存：

tenant_id

department_id

permission_scope

visibility

面试说法：多租户场景下，RAG 必须按 tenant_id 做隔离，避免 A 企业用户检索到 B 企业文档。tenant_id 应该作为 metadata 参与检索过滤，甚至可以按租户拆分 collection。

4. 文档版本和状态过滤

企业制度会更新。

旧制度可能：

status = inactive

version = v1

新制度：

status = active

version = v2

检索时应该优先或只检索：

status = active

否则会出现：

用户问最新报销规则，系统召回旧制度。

面试说法：企业制度文档需要保存 version 和 status。检索时一般过滤 active 文档，避免召回过期制度；如果用户明确问历史版本，再按 version 检索。

5. 工程优化主要看什么？

RAG 工程优化不是只看效果，还要看：

延迟

成本

稳定性

可观测性

缓存

限流

降级策略

常见优化：

1. 减少 Top-N / Top-K
2. 缓存高频 query 的检索结果
3. rerank 只对必要问题启用
4. embedding 批量计算
5. 异步化文档入库
6. 记录 trace_id 和检索日志
7. LLM 超时时降级或拒答

面试说法：RAG 工程优化要在效果、延迟和成本之间平衡。可以通过控制召回数量、缓存高频问题、批量 embedding、选择更小的模型、减少不必要的 rerank、记录 trace_id 和日志来优化。

6. 你的项目怎么讲?

我的项目在 chunk payload 里保存 doc_id、page、section_path、status、version、permission_scope 等 metadata。在线检索时会结合当前用户身份做过滤，避免无权限或过期文档进入候选集。生成前也会检查证据来源，确保 Prompt 中只包含用户可访问的证据。工程上会记录 trace_id、kb_query、检索耗时、Top-K 证据和引用信息，方便排查检索、rerank、生成和权限问题。