

RAG 완전 이해

검색 증강 생성부터

운영·권한·무종단 업데이트 전략까지

```
retriever.search()
```

```
query = "개인 USB 써도 되나요?"
```

```
filter = { is_active: true, role: "member" }
```

```
# 10,000 chunks → Top-K
```

```
0.91 security_policy.md #14
```

```
0.88 security_policy.md #15
```

```
0.74 it_guide.md #03
```

```
0.41 travel_policy.md #07 (cut)
```

```
✓ context 3 chunks · with source
```

Chunking

Embedding

Top-K

Reranking

권한 · 운영

RAG를 기술이 아니라 설계로 이해한다

RAG를 단순히 "PDF를 Vector DB에 넣고 검색하는 기술"로 외우는 것이 아니라, 언제 RAG가 필요한지, 어떤 데이터에 적합한지, 검색 품질과 Latency를 무엇이 결정하는지, 운영 중 문서와 Embedding Model을 어떻게 안전하게 업데이트하는지까지 설계 관점에서 이해한다.

1교시

**RAG가 무엇이고
언제 필요한가?**

RAG 정의, 정형/비정형 데이터, Vector DB, Semantic Search, 기본 RAG 모델

2교시

**검색은 실제로
어떻게 동작하는가?**

Chunking, Embedding, Metadata, Top-K, Threshold, Reranking, Hybrid Search

3교시

**서비스로 만들려면
무엇을 설계해야 하는가?**

Latency, Indexing, 업데이트, 무중단 재임베딩, 권한, 평가·모니터링

1교시 · RAG란 무엇인가, 왜 쓰는가

"AI가 데이터를 써야 한다"는 이유만으로 RAG를 써야 하는가?

아닙니다. 먼저 데이터의 성격과 필요한 조회 방식을 구분해야 합니다.

RAG 정의

정형 / 비정형

Vector DB

Semantic Search

기본 RAG 모델

RAG = Retrieval-Augmented Generation

핵심은 LLM이 답변하기 전에 외부 지식에서 관련 정보를 검색(Retrieval)하고, 그 결과를 Context로 넣어 답변 생성(Generation)을 보장하는 것입니다.



한 문장 정의

RAG는 "모델에게 모든 지식을 다시 학습시키는 기술"이 아니라, "질문할 때 필요한 지식을 찾아서 LLM에게 보여주는 기술"이다.

Retrieval — 필요한 지식을 찾는다

Generation — 그 근거로 답을 만든다

어떤 자료를 RAG에 넣을 수 있는가

RAG의 원본 데이터는 특정 파일 형식으로 제한되지 않습니다. 다양한 자료를 검색 가능한 지식으로 변환할 수 있습니다.

PDF / DOCX / PPTX

규정, 보고서, 매뉴얼, 계약서, 발표자료

XLSX / CSV

표 형태도 가능하지만, 모든 질문에 Vector Search가 적합한 것은 아님

Notion / Confluence / Drive

사내 지식 저장소 · SharePoint 포함

Slack / Teams / 이메일

대화와 업무 맥락

웹페이지 / Wiki / 기술문서

지속적으로 업데이트되는 지식

주의 — "파일을 넣을 수 있다"와 "RAG로 검색하는 것이 최선이다"는 다른 문제다. 데이터가 표 형태로 명확하게 정형화되어 있다면 DB Query나 Function Calling이 더 적절할 수 있다.

RAG는 언제 써야 하는가 — 정형과 비정형

데이터 유형	예시	권장 접근
정형 데이터	직원번호, 잔여연차, 주문금액, 재고수량, 상태코드	DB/SQL, API, Spreadsheet, Function/Tool Calling
비정형 데이터	규정집, 계약서, 매뉴얼, 보고서, 회의록, 기술문서	RAG / Search / Document Retrieval
반정형 데이터	JSON, 로그, 표가 섞인 문서, Wiki	필터-구조 조회 + RAG를 혼합

"김민수의 잔여 연차가 몇 일인가?"



HR DB에 정확한 숫자가 있다면
Vector DB에서 유사 문장을 찾을 이유가 없다

API / SQL Tool 호출

"장기 연차를 사용할 때 어떤 승인 절차가 필요한가?"



규정 문서의 여러 문장을
해석해야 하므로 RAG가 잘 맞는다

RAG

실무 원칙 — 정확한 값과 상태를 조회하는 문제는 정형 DB/API가 우선이다. 의미를 읽고 해석해야 하는 대규모 비정형 지식은 RAG가 강하다. 기업 Agent는 둘을 함께 쓰는 경우가 많다.

같은 질문처럼 보여도 필요한 기술은 다르다

사용자 질문	필요한 소스	적합한 방식
김민수의 남은 연차는?	HR DB	Function Calling → DB/API
5일 연속 연차가 가능한가?	HR 규정 문서	RAG
김민수가 다음 주 5일 휴가 가능한가?	잔여 연차 + 규정 + 일정	DB Tool + RAG + Calendar Tool
A-19382 주문 상태는?	주문 DB	정확 조회
제품 A의 설치 시 주의사항은?	제품 매뉴얼	RAG

Vector DB란 무엇인가

문서를 의미 기반으로 검색하려면 각 문서 조각을 Embedding Vector로 표현하고, 이 Vector를 빠르게 저장·검색할 공간이 필요합니다. Vector Store는 Vector와 원문, Metadata를 함께 관리하고 가까운 Vector를 빠르게 찾습니다.



일반 DB

"값이 정확히 같은가?"

VECTOR DB

"의미가 얼마나 가까운가?"

Vector DB는 Vector · 원문 · Metadata를 함께 보관한다. 이 Metadata가 뒤에서 권한과 버전을 통제하는 수단이 된다.

Semantic Search — 단어가 달라도 의미로 찾는다

사용자 표현	문서 표현	Keyword만 사용	Semantic Search
USB 써도 돼?	승인되지 않은 개인 이동식 저장장치 사용 금지	놓칠 수 있음	의미가 비슷해 검색 가능
남은 휴가	잔여 연차	표현 차이 존재	유사 의미로 연결 가능
PR 승인 몇 명?	Pull Request 최소 1명 승인	약어/정식명칭 차이	문맥 기반 검색 가능

질문과 문서 모두를 Embedding하여 같은 Vector 공간에 놓고, 거리 또는 유사도를 계산하기 때문이다.

RAG의 기본 모델 — 세 단계만 기억하면 충분하다

1

Indexing

문서를 미리 잘라 Embedding하고 검색 가능하게 저장

질문 이전 · 사전 작업

2

Retrieval

질문과 관련 높은 문서 조각을 찾음

질문 시점

3

Generation

검색된 문서를 Context로 넣고 LLM이 답변

질문 시점

조금 더 발전하면

형태	설명	특징
Naive / Basic RAG	질문 → 검색 → Top-K → LLM	구조가 단순하고 빠름
Advanced RAG	Query Rewrite, Hybrid Search, Reranking, Metadata Filter 추가	검색 정확도와 운영성을 개선
Agentic RAG	Agent가 검색 여부·소스·재검색 여부를 판단	유연하지만 비용·Latency·예측 가능성 관리 필요

처음부터 Agentic RAG를 목표로 하지 않는다. Basic RAG가 충분한 문제라면 단순한 구조가 더 빠르고 안정적이다.

RAG가 해결하는 것과 해결하지 못하는 것

RAG가 잘 해결하는 문제

모델이 학습하지 않은 사내 지식

최신 문서와 자주 바뀌는 지식

대규모 문서 중 관련 부분 검색

근거 문서와 출처 제공

LLM 입력 Context를 선택적으로 구성

RAG가 자동으로 해결하지 않는 문제

잘못된 원문 자체의 오류

권한 관리

잘못된 Chunk / Embedding 설계

환각 100% 제거

검색 결과의 최신성 / 버전 충돌

2교시 · RAG 구성요소와 검색 원리

"10,000개의 문서 조각 중 왜 이 3개가 LLM에게 들어갔는가?"

이 질문에 답할 수 있으면 RAG의 핵심을 이해한 것이다.

Parsing

Chunking

Embedding

Metadata

Top-K

Threshold

Reranking

Hybrid Search

RAG 전체 파이프라인

INDEXING · 사전 준비

원본 데이터 수집

Parsing / Cleaning

Chunking

Metadata 부여

Embedding

Vector Index 저장

RETRIEVAL · 질문 처리

Query 처리

Retrieval

Top-K / Threshold

Reranking (선택)

여기서 10,000개가 몇 개로 줄어든다

GENERATION · 답변 구성

Context 구성

LLM Answer + Source

출처 없는 답변은 검증할 수 없다. Source는 옵션이 아니다.

Parsing과 Cleaning

RAG 품질은 Vector DB 이전에 원본 데이터 품질에서 시작합니다. PDF에서 Header/Footer가 반복되거나, 표가 깨지거나, OCR 오류가 섞이면 그 오류까지 검색 대상이 됩니다.

반복 Header / Footer, 페이지 번호 제거

표 · 목록 · Heading 구조 보존

OCR 오류 및 깨진 문자 점검

중복 문서와 폐기 버전 정리

문서 고유 ID와 버전 정보 유지

Garbage In → Garbage Out. 검색 품질이 나쁠 때 Embedding Model부터 바꾸기 전에 "원문이 제대로 읽혔는가?"부터 확인해야 한다.

Chunking — 검색되는 것은 파일이 아니라 Chunk다

CHUNK가 너무 작을 때



문맥이 끊기고 조건·예외가 분리될 수 있음

검색은 정밀하지만 답변에 필요한 주변 정보가 부족

Chunk 수가 증가해 Index와 검색 부담 증가 가능

CHUNK가 너무 클 때



질문과 무관한 내용까지 함께 들어옴

Context Token 증가

한 Chunk가 여러 주제를 포함해 Embedding 의미가 흐려질 수 있음

Chunk Size는 "몇 글자가 정답"이 아니라 문서 구조와 질문 유형에 따라 결정한다.

문서 유형별 자연스러운 Chunk 기준

문서 유형	자연스러운 Chunk 기준
FAQ	질문 + 답변 한 세트
규정집 / 법률	조항 / 항 / 절
Markdown / 기술문서	Heading / Section
코드	함수 / Class / 모듈
회의록	안건 / Topic / 발언 블록
매뉴얼	기능 또는 작업 절차

CHUNK OVERLAP



Chunk 경계에서 의미가 끊기는 문제를 줄이기 위해 앞뒤 Chunk의 일부를 겹치게 할 수 있습니다.

Overlap을 크게 잡으면 Vector 수와 중복 검색 결과가 늘고, 동일한 문장이 여러 번 Context에 들어갈 수 있다.

원칙 — Overlap은 문맥 보존을 위한 안전장치이지, 높을수록 좋은 성능 파라미터가 아니다.

Embedding — 의미를 숫자 Vector로

"USB 사용 가능?"과 "개인 이동식 저장장치 규정"처럼 단어가 달라도 의미가 가까운 문장을 비슷한 위치에 배치하는 것이 목표입니다.

같은 Vector 공간 안의 거리 (개념 예시)

query = "USB 사용 가능?"

0.91 "개인 이동식 저장장치 사용 금지"

0.78 "외부 매체 반입 절차"

0.24 "해외 출장 신청 기한"

차원 수는 모델이 결정한다

dim = 1536 # 모델을 바꾸면 이 값도 바뀔 수 있다

LLM

Embedding Model

생성, 요약, 추론, Tool 선택

텍스트 → Vector

답변 품질에 직접 영향

검색 후보 품질에 직접 영향

Prompt / Context를 입력받음

문서와 Query를 같은 공간에 표현

둘은 다른 모델이다. 검색이 안 될 때 LLM을 바꾸면 아무 일도 일어나지 않는다.

Metadata — 이 Chunk가 무엇인지 설명하는 정보

```
# 하나의 Chunk 레코드
{
  "chunk_id": "security_policy#14",
  "document_id": "security_policy",
  "text": "승인되지 않은 개인 이동식 ...",
  "department": "all",
  "version": "v2",
  "effective_from": "2026-01-01",
  "is_active": true,
  "security_level": "internal",
  "allowed_roles": ["member", "admin"],
  "vector": [0.18, -0.42, ...]
}
```

source / document_id / chunk_id

department / project / customer / tenant_id

year / version / effective_date / updated_at

document_type / language / author

security_level / allowed_roles / allowed_user_ids

Semantic Search만으로 해결하기 어려운 버전, 부서, 권한, 고객, 문서 종류 등을 제어하는 데 매우 중요합니다.

실무 포인트 — 정확히 알고 있는 조건은 Metadata Filter로 제한하고, 의미가 필요한 부분만 Semantic Search에 맡기면 검색 정확도와 보안이 모두 좋아진다.

RAG는 연관성 높은 몇 개만 찾는다



K = 3 일 때

0.91 security_policy #14

0.88 security_policy #15

0.74 it_guide #03

K = 3 컷

0.41 travel_policy #07

0.36 meeting_note #22

Top-K는 "최대 몇 개의 Chunk를 가져올 것인가"를 정한다. 다만 순위만으로 충분하지 않을 수 있다.

Similarity Threshold와 Top-K의 Trade-off

SIMILARITY THRESHOLD

일정 기준보다 관련성이 낮은 검색 결과를 제외하는 조건

주의 — Similarity Score의 범위와 의미는 Vector DB와 거리 메트릭에 따라 다를 수 있으므로 "0.7이 항상 정답"처럼 고정하면 안 된다.

TOP-K TRADE-OFF

K가 너무 작으면

정답에 필요한 정보가 빠진다

K가 너무 크면

Noise · Token · Latency가 증가한다

Noise

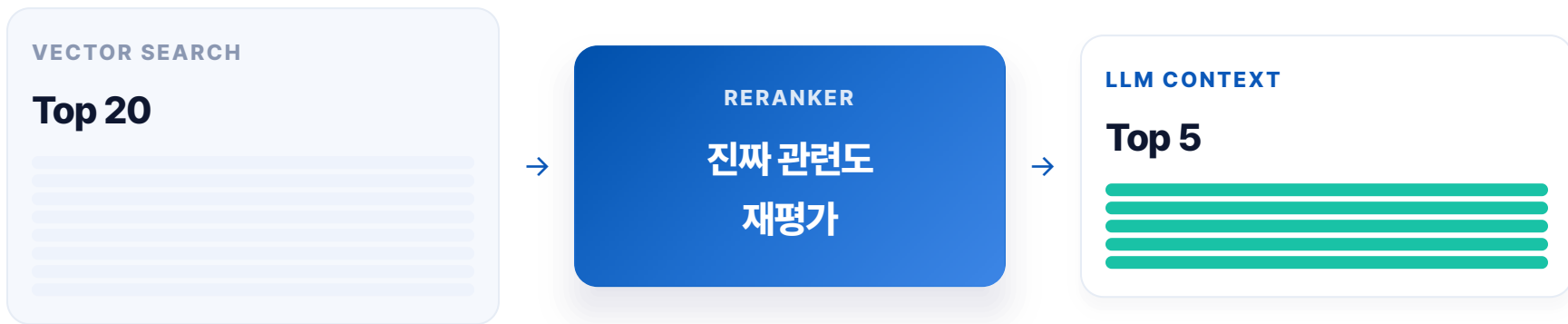
Token 비용

Latency

고정된 정답값은 없다. 실제 질문 세트로 조정한다.

Reranking — 후보를 다시 평가한다

1차 검색 후보를 다시 평가해 실제 관련성이 높은 Chunk를 최종 Context로 고르는 단계입니다.



정확도는 높일 수 있지만 추가 모델 호출이나 계산 때문에 Latency와 비용이 늘 수 있다.

Keyword Search와 Hybrid Search

Vector Search는 의미에는 강하지만 **정확한 코드, ID, 고유명사, 에러번호** 검색에서는 Keyword Search가 더 유리할 수 있습니다.

질문 / 데이터	더 중요한 방식
"로그인 문제 해결법"	Semantic Search
"ERR-DB-18493"	Keyword / Exact Match
"AB-39472 제품 설치 오류"	Keyword + Semantic
사번 / 주문번호 / 정확 조항번호	Structured / Keyword

Hybrid Search — Keyword Search와 Vector Search를 함께 사용해 정확한 용어와 의미적 유사성을 동시에 활용하는 방식

Query Rewrite / Multi-Query

사용자 원문

"그거 회사에서 써도 돼?"

대화 맥락을 이용한 Rewrite

"회사 업무용 PC에서 개인 USB
저장장치 사용 규정"

Multi-Query 확장

q1 = "개인 이동식 저장장치 사용 금지"

q2 = "외부 매체 반입 승인 절차"

q3 = "보안 정책 USB 예외 조항"

→ 결과를 합쳐 후보 구성

사용자가 검색에 적합한 표현으로 질문한다는 보장은 없습니다. 대화 맥락을 이용해 검색에 맞는 표현으로 바꿀 수 있습니다.

하나의 질문을 여러 검색어로 확장해 결과를 합치는 **Multi-Query** 전략도 사용할 수 있습니다.

다만 Query를 늘릴수록 검색 횟수와 Latency가 증가하므로 무조건 적용하는 기능은 아니다.

Context 구성과 출처

Retriever가 찾은 Chunk를 그대로 무작정 붙이는 것도 설계 대상입니다. 중복 제거, Source 순서, 문서 버전, Context 길이 제한, 질문과의 관련성을 고려해 최종 Context를 구성해야 합니다.

Chunk별 출처와 문서 ID 유지

중복 문장 제거 또는 압축

폐기된 버전 제외

충돌 문서는 최신성 / 우선순위 규칙 적용

최종 답변에서 Source Citation 제공

RAG를 "한 번 만들어보는 데모"가 아니라 "문서가 계속 바뀌고 사용자가 동시에 접속하는 실제 서비스"로 생각한다.

LATENCY는 전체 경로의 합이다

RAG의 응답속도는 LLM 하나만의 문제가 아닙니다. 각 단계가 100~500ms씩만 추가되어도 전체 응답은 빠르게 느려집니다.

RAG에는 Latency Budget이 필요하다



Latency를 낮추는 전략

1 문서 Embedding은 Query 시점이 아니라 Indexing 시점에 미리 생성한다

2 Metadata Pre-filter로 검색 대상 Vector 수를 줄인다

3 Top-K 후보 수를 목적에 맞게 제한한다

4 Reranking은 필요한 Query에만 적용하거나 후보 수를 제한한다

5 자주 반복되는 Query / 검색 결과는 Cache를 고려한다

6 여러 데이터 소스를 조회해야 한다면 가능한 부분을 병렬 처리한다

7 불필요한 Query Rewrite / Multi-Query를 항상 수행하지 않는다

8 Vector Index 구조와 Search Parameter를 데이터 규모에 맞게 튜닝한다

정확도를 높이기 위해 단계를 계속 추가하면 Latency가 커진다. RAG 설계는 Accuracy – Cost – Latency의 균형 문제다.

인덱싱 전략

서비스가 커지면 "전체 문서를 매번 처음부터 다시 Embedding"하는 구조는 비용과 시간이 커집니다.

전략	설명
Full Re-index	전체 문서를 다시 Chunk / Embedding. 초기 구축이나 대규모 정책 변경에 단순함
Incremental Indexing	새로 추가·수정된 문서만 처리
Change Detection	문서 Hash / updated_at / version을 비교해 변경분 감지
Upsert	동일 document_id / chunk_id의 Vector를 갱신
Delete / Tombstone	삭제되거나 폐기된 문서가 검색되지 않도록 제거 / 비활성화

문서가 업데이트되면 어떻게 할 것인가

원본 문서 변경 감지



새로운 Version / Effective Date 확인



변경된 Chunk 재생성 → Embedding



새 Index에 Upsert



구버전 비활성화 / 삭제



검색 품질 검증

문제 상황

설명서 v1

+

설명서 v2

→ 둘 다 검색

단순히 v2를 추가만 하면 v1과 v2가 모두 검색되어 충돌할 수 있습니다.

SOURCE OF TRUTH

Vector DB를 원본 저장소처럼 취급하지 않는다. 원본 문서 시스템이 Source of Truth이고, Vector Index는 언제나 재생성 가능한 검색용 파생 데이터다.

버전 관리 전략

document_id 같은 문서의 정체성

version v1, v2 또는 해시

effective_from / to 실제 유효 기간

is_active 현재 검색 대상인지

updated_at 변경 감지와 감사

index_version 어느 인덱싱 파이프라인으로 생성되었는지

최신 버전만 검색하도록 강제

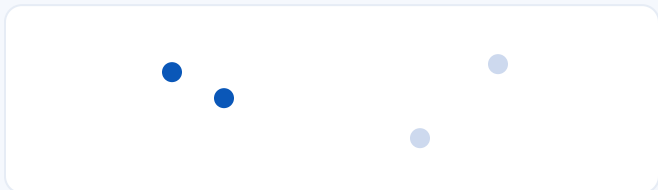
```
{
  "filter": {
    "is_active": true,
    "version": "current",
    "effective_from": { "$lte": "today" }
  },
  "top_k": 5
}
```

✓ 폐기된 v1은 후보에 들어오지 않는다

최신 버전만 검색해야 하는 서비스라면 Metadata Filter로 이런 조건을 강제할 수 있습니다.

Embedding Model을 바꾸면 어떤 문제가 생기는가

MODEL A로 만든 Vector

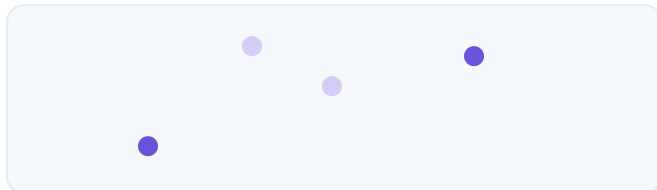


dim = 1536

같은 공간이 아니다



MODEL B로 만든 Query Vector



dim = 3072 (달라질 수도 있다)

Embedding Model 교체는 코드 한 줄 변경이 아니라 "기존 Corpus를 새 모델로 다시 Embedding해 새 Index를 구축하는 마이그레이션"이다.

서비스 중단 없이 Embedding Model을 교체하는 전략

현재 서비스 → Index V1 (Embedding Model A)

백그라운드 → 전체 Corpus → Index V2 (Model B)

V2 품질 / Latency 검증

일부 Traffic Shadow / Canary 비교

검색 Alias / Pointer를 V2로 전환

문제 없으면 V1 보관 후 제거

기존 Index를 삭제하고 새 모델로 전체 Embedding하는 동안 서비스를 멈추는 방식은 피해야 합니다. Blue-Green 방식처럼 두 Index를 병렬로 운영할 수 있습니다.

ROLLBACK

새 모델의 품질이 예상보다 나쁘면 Alias를 다시 V1으로 돌릴 수 있어야 한다. 그래서 "덮어쓰기"보다 "새 인덱스 생성 → 전환"이 운영상 안전하다.

Embedding Model 교체 시 체크할 항목

1 Vector 차원(Dimension)과 DB Index 설정 호환성

2 한국어 / 영어 / 다국어 검색 품질

3 도메인 용어와 약어 검색 품질

4 기존 Test Query의 Recall@K 변화

5 Index Size와 Embedding 비용

6 Query Embedding Latency

7 Reranker와의 조합 효과

8 새 모델로 재임베딩할 전체 Corpus 시간과 비용

RAG 권한 조정은 반드시 필요하다

RAG가 권한을 자동으로 관리하는 것은 아닙니다. 사용자 권한을 확인한 뒤 **Retrieval 단계에서 검색 범위를 강제로 제한**해야 합니다.

사용자 로그인



Identity / Role / Tenant 확인



허용 범위 Filter 생성



권한 Filter를 적용한 Retrieval



허용된 Chunk만 LLM Context에 전달

사용자 Identity → 검색 Filter

```
user = { role: "member", dept: "sales" }
```

```
filter = {
```

```
  "tenant_id": "acme",
```

```
  "allowed_roles": { "$contains": "member" },
```

```
  "security_level": { "$in": ["public", "internal"] }
```

```
}
```

× confidential 문서는 후보에도 없다

보안 원칙 — 권한 없는 문서는 "검색은 했지만 답변에서 숨긴다"가 아니라, 애초에 Retrieval 결과와 LLM Context에 들어가지 않게 해야 한다.

권한 Metadata 설계와 멀티테넌트 RAG

권한 METADATA

tenant_id

department

security_level

allowed_roles

allowed_user_ids

권한 Filter는 유사도보다 우선하는 강제 조건이다.

멀티테넌트 RAG · 두 가지 선택

공유 Index + 강제 Filter

모든 Query에 tenant_id 필터를 강제한다. 운영은 단순하지만 필터 누락이 곧 사고다.

단순

누락 위험

Tenant별 Collection / Index 분리

보안 요구가 높으면 물리적으로 격리한다.

격리

운영 부담 증가

RAG 품질은 어떻게 평가하는가

최소한 **Retrieval**과 **Generation**을 분리해서 평가해야 합니다.

평가 영역	질문
Retrieval Quality	정답이 있는 Chunk 를 Top-K 안에 가져왔는가?
Relevance	가져온 Chunk 가 실제 질문과 관련 있는가?
Groundedness / Faithfulness	답변이 검색된 Context 에 근거하는가?
Answer Correctness	최종 답이 사용자의 질문에 맞는가?
Citation Quality	출처가 실제 근거와 연결되는가?
Latency	p50 / p95 응답시간이 서비스 목표 안에 있는가?

Test Dataset을 만들어야 한다

RAG 튜닝에서는 실제 사용자 질문과 기대 Source를 모은 평가 세트가 필요합니다.

질문	기대 Source	핵심 정답
개인 USB 사용 가능한가?	security_policy	승인 없는 개인 저장장치 금지
본인 결혼 휴가는?	hr_policy	5일
main에 직접 push 가능한가?	development_guide	직접 Push 금지
해외 출장은 언제 신청?	travel_policy	정책상 기한 확인

Chunk Size, Embedding Model, Top-K, Threshold, Reranker를 바꾼 뒤 동일 Test Dataset으로 비교하면 "감"이 아니라 데이터로 개선 여부를 판단할 수 있다.

운영 모니터링 — RAG Observability

검색 결과가 0개인 Query 비율

Top-1 / Top-K Score 분포

사용자가 재질문하는 비율

Source 없는 답변 비율

Retrieval / Rerank / LLM Latency

p50 / p95 / p99 전체 응답시간

평균 Context Token과 비용

오래된 Document Version 검색 비율

RAG는 데이터와 사용자 질문이 계속 변하므로 "한 번 튜닝하면 끝"이 아니다. 운영 중에도 검색 품질과 Latency의 변화를 관찰해야 한다.

성능을 높일 때 생각하는 순서

1 원본 데이터가 깨끗한가?

2 올바른 Chunk 단위인가?

3 Metadata / 권한 / 버전 Filter가 맞는가?

4 Embedding 검색이 정답 후보를 찾는가?

5 Top-K / Threshold가 적절한가?

6 Keyword / Hybrid가 필요한가?

7 Reranker가 필요한가?

8 **마지막에 Generation Prompt / LLM을 점검**

RAG 답변이 이상하다고 바로 더 비싼 LLM으로 바꾸지 않는다. 먼저 Retrieval 단계에서 정답 Context가 실제로 들어왔는지 확인한다.

3시간 전체 정리

RAG를 한 장으로

INDEXING

원본 지식 · PDF / DOCX / Wiki / Slack / Manual



Parsing / Cleaning



Chunking + Metadata



Embedding + Vector Index

QUERY TIME

사용자 Query



Permission / Metadata Filter



Semantic / Keyword / Hybrid Retrieval



Top-K + Threshold → Reranking (필요 시)



Context 구성 + Source



LLM 답변

반드시 기억할 핵심 12가지

1 RAG는 Retrieval + Generation이며 Vector DB 자체를 뜻하지 않는다.

2 RAG는 특히 대규모 비정형·반정형 지식에 강하다.

3 정형 데이터의 정확한 값 조회는 DB/API/Function Calling이 더 적합한 경우가 많다.

4 Semantic Search는 단어가 아니라 의미가 가까운 정보를 찾는다.

5 실제 검색 단위는 문서 전체가 아니라 Chunk인 경우가 많다.

6 Chunk Size, Overlap, Top-K에는 절대적인 정답이 없다.

7 Top-K만 보지 말고 Threshold, Reranking, Hybrid Search를 필요에 따라 조합한다.

8 Metadata는 버전·부서·고객·권한·최신성을 통제하는 핵심 구성요소다.

9 문서 업데이트는 Incremental Indexing과 Version 관리 전략이 필요하다.

10 Embedding Model 교체는 새 Index를 병렬 생성하고 전환하는 마이그레이션 문제다.

11 권한은 LLM 답변 단계가 아니라 Retrieval 이전/동시에 강제해야 한다.

12 RAG 품질은 Retrieval Quality, Answer Quality, Latency를 분리해 측정해야 한다.

좋은 RAG는 "Vector DB를 붙이는 것"이 아니라, 올바른 사용자가 올바른 시점에 올바른 정보를 빠르게 찾아 LLM Context에 넣도록 설계하는 것이다.