생성형 AI 기반 RAG 개발자 과정

RAG 개요 + 문서 전처리

Session 6 / 33 — Day 1

왜 RAG 인가 — 환각을 근거로 잡는 법

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

이번 시간을 마치면

🤔 왜 RAG

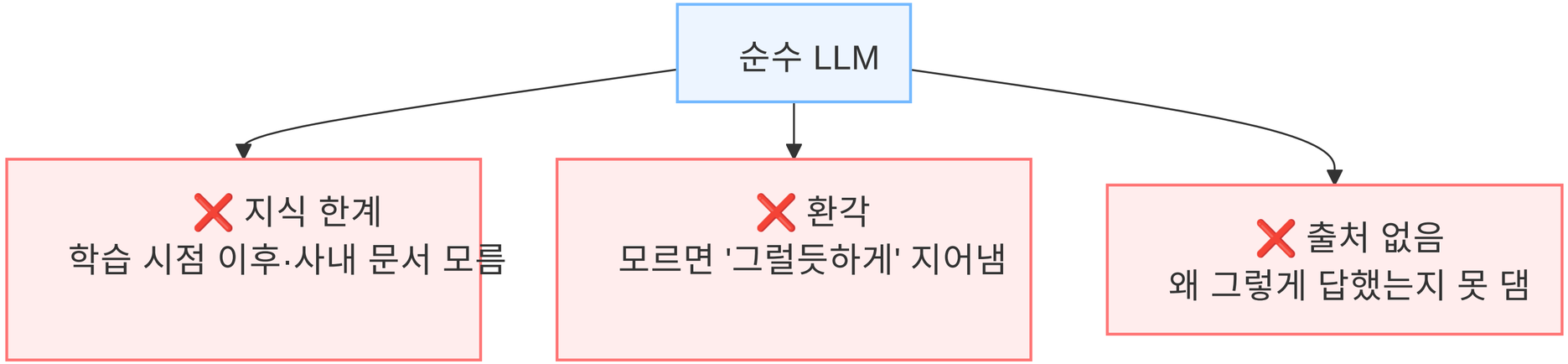
- LLM 의 **3대 한계** (지식 한계·환각·출처 없음)
- **RAG vs Fine-Tuning** 언제 무엇을
- RAG **3단계** 큰 그림

📄 문서 전처리

- 원문 → **텍스트 추출**
- **Chunking** 이 왜 필요한가
- **Chunk Size** 전략

> 🎯 목표: RAG 의 "왜"(필요성)와 "준비"(전처리)를 개념으로 잡는다.

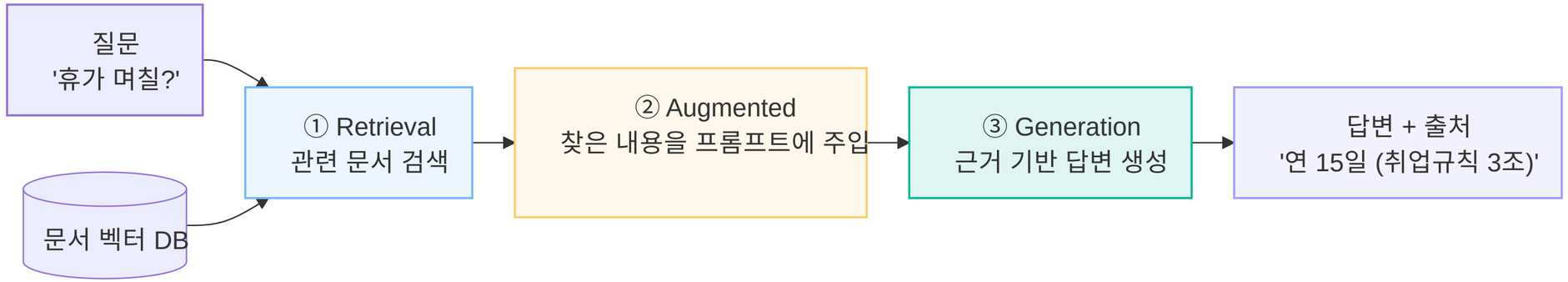
LLM 만으로는 안 되는 이유



- **지식 한계**: "우리 회사 휴가 규정?" → 모델은 모름
- **환각**: 모르면서 자신 있게 틀린 답
- **출처 없음**: 근거 추적 불가 → 업무에 못 씀

> 🔑 이 셋을 **외부 문서 검색**으로 한 번에 해결하는 패턴 = **RAG**.

RAG = 검색 + 생성



💡 RAG = **R**etrieval-**A**ugmented **G**eneration. "외워서 답하지 말고, **찾아보고 답해라**."

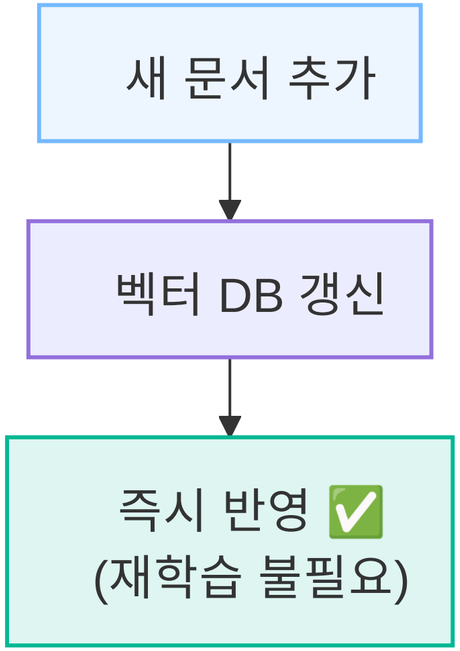
오픈북 시험 vs 암기 시험

순수 LLM = 암기 시험

- 머릿속 지식만으로 답
- 모르면 **지어냄**(환각)
- 새 정보 반영 = **재학습** 필요

RAG = 오픈북 시험

- 질문 받으면 **자료를 펴서** 답
- 자료에 없으면 "모른다" 가능
- 자료만 갈아끼우면 **즉시 최신화**



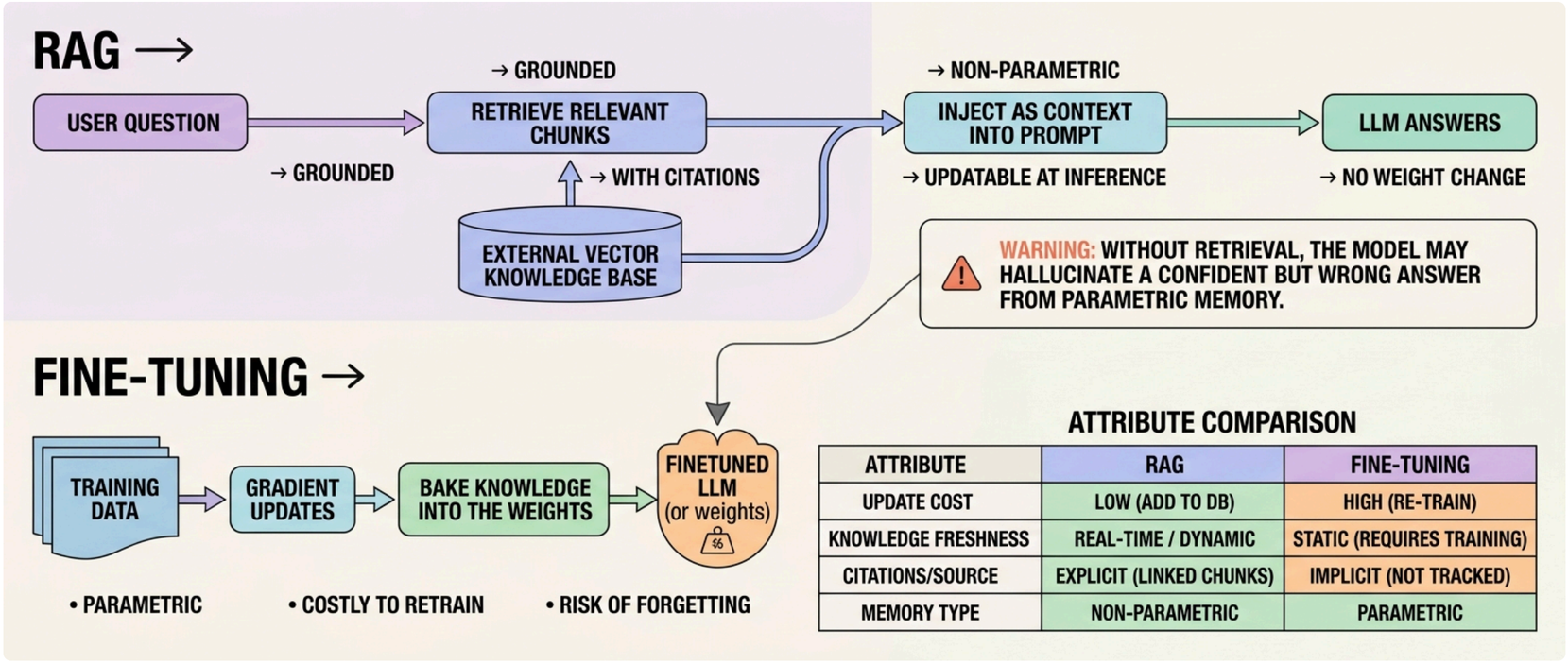
🔑 그래서 **사내 문서·자주 바뀌는 지식**엔 RAG 가 압도적.

언제 RAG, 언제 Fine-Tuning

기준	RAG	Fine-Tuning
하는 일	**지식**을 외부에서 주입	**행동·말투·형식**을 모델에 각인
데이터 변경	문서만 교체 → 즉시	다시 학습 필요
비용	낮음 (검색+프롬프트)	높음 (GPU 학습)
출처 제시	✅ 가능	❌ 어려움
최신성	✅ 강함	❌ 학습 시점 고정
적합	사내 문서 QA, 최신 정보	특정 톤·도메인 문체·출력 형식

> 💡 실무 정답은 보통 **"RAG 먼저, 필요하면 Fine-Tuning 보강"**. 이 코스는 **RAG 에 집중**.

RAG vs Fine-Tuning — 한 그림으로



RAG vs 파인튜닝 · RAG는 외부 지식을 검색해 프롬프트에 주입(갱신 쉬움·출처), 파인튜닝은 가중치에 각인(재학습 비용)

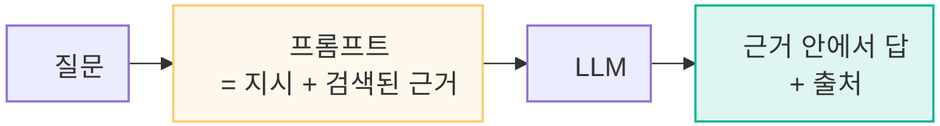
근거를 주면 덜 지어낸다

환각의 뿌리 (복습)

- 모델은 **'그럴듯한 다음 토큰'**을 고름
- 근거가 없으면 빈칸을 **상상**으로 채움

RAG 의 처방

- 프롬프트에 **실제 문서 조각**을 넣음
- "아래 **자료에 근거해서만** 답하라"
- 자료에 없으면 **"모른다"** 하도록 지시



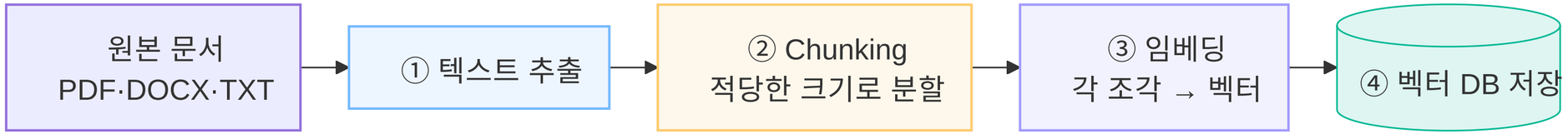
> ⚠️ 주의: RAG 도 환각을 **0 으로** 만들진 못함 → Day2 에서 **출처 표시·검증**까지 (Session 16).

PART B

PART B
문서 전처리 — 검색 잘 되는 자료 만들기

텍스트 추출 · Chunking · Chunk Size — 약 14분

문서가 검색 가능해지기까지



- 이번 세션은 **①·②(추출·Chunking)** 개념에 집중
- **③ 임베딩**과 **④ 저장·검색**은 이후 실습에서 코드로 구현
- 이 4단계가 모든 RAG 서비스의 **문서 색인 파이프라인**

> 💡 어떤 RAG 시스템이든 문서를 받으면 이 순서(추출→분할→임베딩→저장)로 처리한다.

원문에서 글자만 뽑아내기

형식별 추출

- **TXT** — 그대로 읽기 (가장 쉬움)
- **PDF** — 페이지 단위 추출 (레이아웃·표 주의)
- **DOCX** — 단락 단위 추출
- 이미지 PDF → **OCR** 필요(별도)

코스에서 쓰는 도구 (개념만)

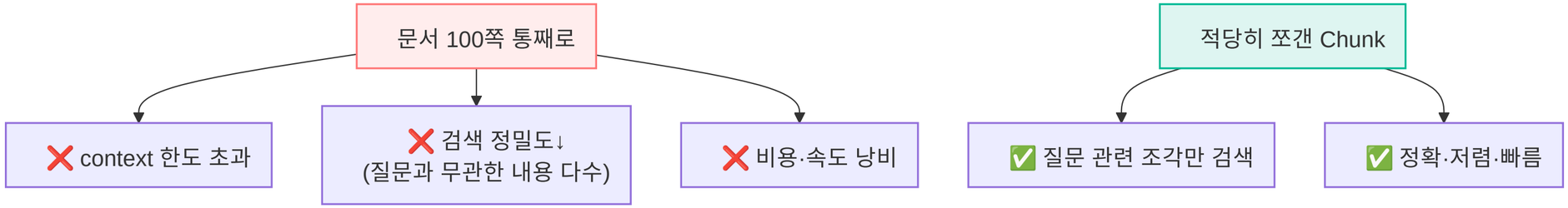
- LangChain **Document Loader** 가 형식별 처리 담당
- 결과물: `Document(page_content, metadata)`

메타데이터가 핵심

- 추출하면서 **출처 정보**를 함께 보관
- `source`(파일명), `page`(쪽), `chunk_id`
- 나중에 **출처 표시·삭제**에 필수

> 🔑 Day2 Session 11 에서 TXT/PDF/DOCX Loader 를 실제로 다룸.
> 코드 경로: `2.langchain/7.RAG/2.loaders/`

문서를 통째로 넣으면 안 되는 이유



- **Chunking** = 문서를 **검색 단위**로 분할
- 너무 크면 → 잡음 많고 한도 초과
- 너무 작으면 → 문맥이 끊겨 의미 손상

> 🔑 "**질문 하나에 답할 만큼**"이 좋은 chunk 크기의 직관.

크기와 겹침, 어떻게 정하나

두 손잡이

- **chunk size** — 한 조각의 길이 (예: 500~1000자)
- **overlap** — 인접 조각 간 겹침 (예: 10~20%)
- 문장이 경계에서 잘려 의미 끊기는 것 방지

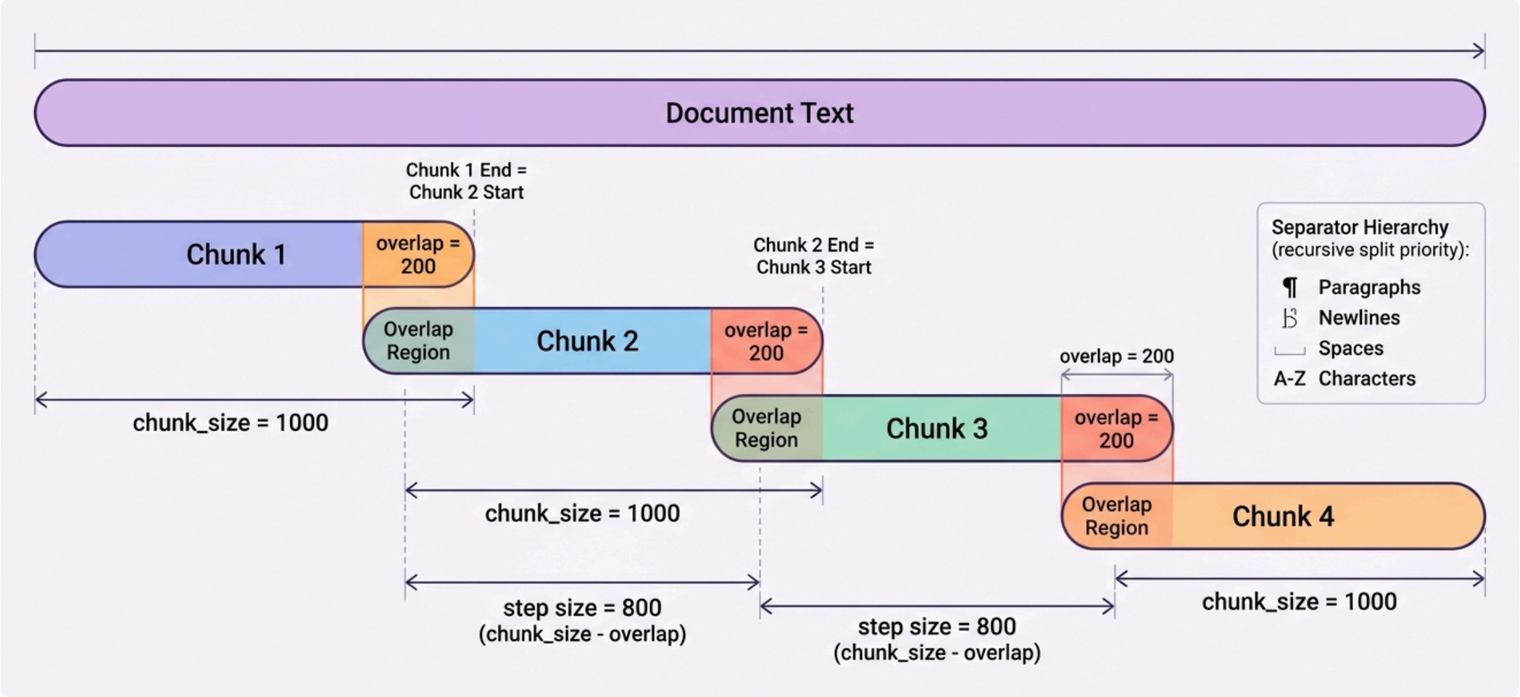
분할 방식 (Day2 상세)

- **Character** — 글자 수로 단순 분할
- **Recursive** — 문단→문장→단어 순으로 자연 분할 (권장)

문서 성격	권장 chunk
매뉴얼·규정	작게 (400~600) — 정밀
서술형·논문	크게 (800~1200) — 문맥
대화·FAQ	항목 단위

> 💡 정답은 없음 → **실험으로 조정**. Day2 Session 12 에서 직접 튜닝.
> 코드 경로: `2.langchain/7.RAG/2.loaders/2.3_chunking.py`

chunk size·overlap — 한 그림으로



청킹 · chunk_size/overlap 슬라이딩 윈도우. 이전 청크 끝을 다음 청크가 일부 겹쳐 경계 문맥 손실 방지

핵심 6가지

1. ✅ LLM 3대 한계: **지식 한계 · 환각 · 출처 없음**
2. ✅ RAG = **검색(R) + 주입(A) + 생성(G)** — "찾아보고 답하기"
3. ✅ **RAG vs Fine-Tuning** — 지식은 RAG, 행동·형식은 FT, 보통 RAG 먼저
4. ✅ RAG 는 근거 주입으로 **환각↓**(0 은 아님 → 출처·검증 필요)
5. ✅ 인덱싱: **추출 → Chunking → 임베딩 → 저장**
6. ✅ Chunk: **작게=정밀 / 크게=문맥**, overlap 으로 경계 보완