고유 데이터 기반 생성 시스템 설계 보고서

검색 증강 생성(RAG)을 넘어서 — 내 데이터를 제대로 찾고, 제대로 답하게 만드는 법

작성일: 2026년 10월 23일 · **개정일:** 2026년 10월 24일 **주제:** 개인·조직의 고유한 자료를 바탕으로 답하는 인공지능(AI) 시스템을 어떻게 설계할 것인가

이 보고서는 기술을 잘 모르는 분도 끝까지 읽을 수 있게 썼습니다. 어려운 용어가 처음 나올 때는 짧은 설명을 붙이고, 용어는 한국어(English) 형태로 함께 적었습니다. 전문 용어를 몰라도, 비유와 예시만 따라오면 전체 그림을 이해할 수 있습니다.

> 한 줄로 요약하면 이렇습니다. **답을 잘하는 비결은 똑똑한 AI 모델이 아니라, 질문에 맞는 증거를 잘 찾아서 모델 앞에 올려놓는 데 있습니다.** 그래서 이 보고서의 중심은 '답을 만드는 법'이 아니라 '자료를 정리하고, 찾고, 검증하는 법'입니다.

이 보고서를 읽는 법

전체 구조는 도서관에 비유하면 이해가 쉽습니다.

- 자료를 잘게 나누는 일을 **청킹(Chunking)**이라고 합니다. 책을 검색하기 좋게 쪽 단위로 나눈 것이라고 생각하면 됩니다.
- 나눈 조각에 '의미의 좌표'를 붙여 컴퓨터가 비슷한 내용을 찾을 수 있게 만드는 일을 **임베딩(Embedding)**이라고 합니다. 책마다 주제 분류 번호를 붙이는 일과 비슷합니다.
- 질문이 들어오면 관련 조각을 찾아오는 일을 **검색(Retrieval)**이라고 합니다. 사서가 서가에서 관련 책을 꺼내 오는 일입니다.
- 찾아온 조각을 읽고 답을 쓰는 일이 **생성(Generation)**입니다. 꺼내 온 책을 펼쳐 보고 보고서를 쓰는 일입니다.

이 네 단계를 합친 가장 기본적인 방식을 **검색 증강 생성(Retrieval-Augmented Generation, RAG)**이라고 부릅니다. '검색으로 재료를 보강한 뒤 답을 생성한다'는 뜻입니다. 문제는 이 네 단계 중 하나라도 부실하면, 마지막 답이 통째로 틀어진다는 점입니다.

자주 나오는 용어, 먼저 풀어 둡니다

용어	쉬운 설명
토큰(Token)	AI가 글을 읽는 최소 단위. 대략 단어 조각 하나. 한국어 한 글자는 보통 1~2개 토큰에 해당합니다. 비용과 분량은 토큰 수로 셉니다.

용어	쉬운 설명
청크(Chunk)	긴 문서를 검색하기 좋게 나눈 작은 조각 하나
벡터(Vector)	글의 의미를 숫자들의 배열로 바꾼 것. 의미가 비슷하면 숫자 배열도 비슷해집니다.
벡터 데이터베이스(Vector Database)	벡터를 저장해 두고 '의미가 비슷한 글'을 빠르게 찾아 주는 저장소
키워드 검색(BM25)	단어가 실제로 등장하는지를 따지는 전통적인 검색 방식. 이름·제품 코드처럼 정확한 단어가 중요할 때 강합니다. 이름 BM25는 대표 알고리즘의 이름입니다.
하이브리드 검색(Hybrid Search)	의미 검색(벡터)과 키워드 검색(BM25)을 함께 쓰는 방식
재정렬 / 리랭킹(Reranking)	1차로 넓게 찾은 후보를, 질문과 하나씩 대조해 순위를 다시 매기는 일. 사서가 꺼내 온 책 100권 중 진짜 필요한 20권만 골라 주는 단계입니다.
지식 그래프(Knowledge Graph)	사람·회사·사건 같은 대상과 그 사이의 관계(소속, 계약, 원인 등)를 점과 선으로 그려 둔 지도
개체 / 엔티티(Entity)	그래프의 점이 되는 대상. 인물, 조직, 제품, 개념 등
다중 홉(Multi-hop)	'A의 담당자가 전에 맡은 고객사는?'처럼 사실 여러 개를 차례로 연결해야 답이 나오는 질문 방식
미세 조정 / 파인튜닝(Fine-tuning)	예시를 많이 보여 줘 AI의 말투·형식·판단 습관을 가르치는 훈련
긴 문맥 / 롱 컨텍스트(Long Context)	자료 전체를 한 번에 AI에게 통째로 건네 읽게 하는 방식
에이전트(Agent)	한 번에 답하지 않고, 도구를 골라 쓰고 결과를 보고 다음 행동을 정하는 AI 작업 방식
환각(Hallucination)	AI가 근거 없이 그럴듯한 내용을 지어내는 현상
출처 표시 / 인용(Citation)	답의 각 주장이 어느 문서의 어느 부분에서 왔는지 밝히는 것

나머지 용어는 나올 때마다 그 자리에서 풀어 설명합니다.

핵심 결론 — 먼저 읽는 요약

방법론의 이름부터 고르지 마세요. 내 자료가 어떤 모양이고, 사람들이 어떤 질문을 던지는지부터 확인하면 답은 대부분 정해집니다.

1. 자료가 작으면, 검색 시스템 자체를 만들지 않는 것이 정답일 수 있습니다. 전체 자료가 약 20만 토큰(Token) — 책으로 치면 약 500쪽 — 이하이고 자주 바뀌지 않는다면, 자료 전체를 AI에게 통째로 건네는 긴 문맥(Long Context) 방식이 더 단순하고, 더 싸고, 종종 더 정확합니다.

2. 문서에서 답을 찾는 질문이 대부분이라면, 잘 만든 기본형으로 충분합니다. 의미 검색과 키워드 검색을 함께 쓰는 하이브리드 검색(Hybrid Search)에, 조각마다 문맥 설명을 붙이는 문맥 기반 검색(Contextual Retrieval)과 재정렬(Reranking)을 더한 조합이 현재 가장 검증된 기본값입니다. 앤트로픽(Anthropic)의 실험에서 이 조합은 '정답 조각을 상위 20개 안에 찾지 못하는 비율'을 5.7%에서 1.9%로, 즉 67% 낮췄습니다.

3. 여러 문서를 연결해야 답이 되는 질문이 많다면, 조각 검색만으로는 부족합니다. 대상과 관계를 지도로 그려 두는 그래프 검색 증강 생성(GraphRAG)이나, 지식을 사람이 읽을 수 있는 문서로 미리 정리해 두는 엘엘엠 위키(LLM Wiki)가 필요합니다. 단, 그래프는 단순한 사실 질문에서는 오히려 기본 방식보다 못할 수 있다는 연구 결과가 있습니다. '연결'이 핵심인 질문에만 쓰는 것이 원칙입니다.

4. 말투·형식·판단 방식이 문제라면, 그건 검색 문제가 아닙니다. 미세 조정(Fine-tuning)은 AI의 행동을 가르치는 도구입니다. 자주 바뀌는 사실을 외우게 하는 데 쓰면, 낡은 지식을 고칠 수도, 출처를 댈 수도 없는 상태가 됩니다. 사실은 검색으로, 행동은 훈련으로 — 이 구분이 중요합니다.

5. 어떤 방법을 쓰든, 측정 없이는 개선이 없습니다. 실제 질문 50~100개와 그 정답을 먼저 모아 두는 평가 집합(Evaluation Set)이 출발점입니다. 감으로 조각 크기를 바꾸는 것은 도박과 같습니다.

여기에 두 자료에서 새로 확인한 점을 더합니다.

- **길게 넣으면 흐려집니다.** 입력이 길어질수록 AI가 내용을 놓치는 현상인 문맥 부패(Context Rot)가 모든 최신 모델에서 확인됐습니다. '들어간다'와 '잘 찾는다'는 다릅니다.
- **미리 정리할 것인가, 질문할 때 찾을 것인가.** 모든 방법론의 차이는 결국 이 한 가지 — 정리 비용을 언제 치를 것인가 — 로 모입니다.
- **숫자 질문은 검색으로 풀지 않습니다.** '지난 분기 계약은 몇 건인가' 같은 세고 더하는 질문은 데이터베이스 질의(SQL)가 담당해야 합니다.

---

1. 왜 검색 증강 생성(RAG)은 생각보다 쉽게 실패하는가

1.1 겉보기에는 간단해 보이는 이유

인터넷 강좌에서 소개하는 기본 형태는 이렇습니다.

```
문서 모으기 → 일정 크기로 자르기(청킹) → 의미 좌표 붙이기(임베딩)
→ 저장소에 넣기 → 질문과 비슷한 조각 찾기(검색) → AI에게 주고 답 쓰기(생성)
```

하루면 시범용으로 만들 수 있습니다. 그런데 실제 업무에 붙이면 답이 자주 어긋납니다. 이유는 단순합니다. 이 방식은 하나의 기술이 아니라 여섯 단계가 이어진 작업 흐름(파이프라인, Pipeline)이고, **최종 품질은 가장 약한 단계가 결정**하기 때문입니다.

1.2 단계마다 무엇이 깨지는가

단계	무엇이 깨지는가	사용자가 보는 증상
수집·해석(Parsing)	표, 여러 단으로 된 지면, 스캔한 그림, 도표가 글자로 옮기는 과정에서 망가짐	표의 숫자가 엉뚱한 항목과 섞여 답이 됨
나누기(Chunking)	조각이 원래 맥락(누가, 언제, 어느 문서)을 잃음. 표와 목록이 중간에서 잘림	'매출이 3% 늘었다'는 조각만 남아 어느 회사·어느 분기인지 모름
의미 검색 (Embedding Search)	'의미가 비슷한 글'과 '정답이 든 글'은 다름. 이름·코드·숫자에 약함	오류 코드 TS-999를 물었는데 비슷한 다른 코드 문서가 나옴
순서 매기기(Ranking)	가져올 개수를 고정하면 정답 조각이 11번째에서 잘림	문서에 답이 있는데 '정보가 없다'고 답함
답 쓰기(Generation)	찾아온 근거와 AI가 원래 알던 내용이 충돌함	출처와 다른 숫자, 그럴듯한 지어낸 답(환각)
운영(Operation)	문서가 바뀌거나 지워졌는데 검색 쪽에 반영이 안 됨. 볼 권한이 없는 문서가 섞임	폐기된 옛 규정으로 답함, 권한 없는 문서가 노출됨

1.3 문맥 손실(Context Loss) — 가장 크고 가장 과소평가되는 문제

조각(청크)은 검색의 단위입니다. 경계를 잘못 자르면, 그 정보는 어떤 좋은 검색 기술을 써도 찾을 수 없게 됩니다. 문맥이 사라지는 방식은 여섯 가지로 나눌 수 있습니다.

사라지는 것	예시	결과
가리키는 대상	'이 회사', '해당 기술', '그 계획'	무엇에 대한 얘기인지 알 수 없음
글의 구조	제목과 본문, 본문과 부록이 떨어짐	그 글이 문서에서 어떤 역할을 하는지 모름
표의 맥락	데이터 행과 열 제목이 떨어짐	숫자가 엉뚱한 항목에 붙음
원인과 결과의 관계	원인은 앞 조각, 결과는 뒤 조각	인과관계를 찾지 못함
시간	옛 계획과 바뀐 계획이 떨어짐	오래된 정보로 답함
논리의 연결	주장과 근거, 예외 조건이 떨어짐	단정적이거나 틀린 답이 나옴

계약서를 예로 들어 보겠습니다. 4조에는 '고객은 매달 일정 금액을 낸다'고 돼 있고, 9조에는 '예외적으로 환불되는 조건'이 있습니다. '환불이 되나요?'라는 질문에 4조만 찾아오면 잘못된 결론이 나옵니다. **조각 하나를 찾았다는 것과, 답에 필요한 증거를 충분히 찾았다는 것은 전혀 다른 일입니다.**

점검법은 간단합니다. **사람이 그 조각만 읽고 무슨 말인지 알 수 없으면, AI도 알 수 없습니다.** 이 한 문장 검사가 나누기 품질을 가르는 가장 빠른 방법입니다.

1.4 질문의 종류 자체가 안 맞는 경우도 있습니다

기본 검색 방식은 '답이 문서 어딘가 한 곳에 들어 있는' 질문에 강합니다. 다음 세 종류는 구조적으로 약합니다.

- **전역 질문(Global Question)** — '이 자료 전체에서 반복되는 주제는?'처럼 모든 문서를 훑어야 하는 질문. 조각 몇 개만 가져오는 구조로는 답할 수 없습니다.
- **다중 홉 질문(Multi-hop Question)** — 'A 프로젝트 담당자가 이전에 맡았던 고객사는?'처럼 사실을 여러 번 연결해야 하는 질문.
- **집계·계산 질문(Aggregation Question)** — '지난 분기 계약은 몇 건인가?'처럼 세거나 합산해야 하는 질문. 이것은 검색이 아니라 데이터베이스 질의의 영역입니다.

> 실패를 'AI 모델이 부족해서'라고 진단하면 해결책이 모델 교체밖에 남지 않습니다. 실제로는 대부분의 실패가 모델이 답을 쓰기 훨씬 전 — 자료를 해석하고, 나누고, 저장하는 단계 — 에서 이미 정해져 있습니다.

---

2. 설계 전에 먼저 답해야 할 여덟 가지 질문

방법론을 고르기 전에, 내 자료와 내 질문을 먼저 살펴봐야 합니다. 아래 여덟 가지에 답하면 구조는 대부분 저절로 좁혀집니다.

확인할 것	무엇을 보는가	구조에 미치는 영향
자료의 크기	전체 분량을 토큰으로 바꾸면 얼마인가	수십만 토큰 이하면 통째로 넣는 방식이 가장 정확할 수 있음. 수백만 토큰 이상이면 검색 층이 필수
자료의 모양	줄글, 표·데이터베이스, PDF·발표 자료, 프로그램 코드, 대화 기록 중 무엇이 중심인가	표는 데이터베이스 질의, 시각 문서는 그림 기반 검색, 코드는 파일 탐색이 유리
질문의 종류	한 곳에서 찾는 사실, 전체 요약, 연결 추적, 세기·합산의 비율	사실 위주면 기본 고도화로 충분. 연결·전역 질문이 많으면 그래프나 사전 정리 층이 필요
바뀌는 속도	한 번 쌓고 끝인가, 매주 바뀌는가, 실시간인가	자주 바뀌면 다시 만드는 비용이 큰 그래프 방식이 불리. 실시간이면 원본을 직접 읽는 방식이 유리
정확도 기준	틀리면 얼마나 큰 문제인가, 출처 제시가 필수인가	높을수록 출처 강제, 재정렬, 답-근거 일치 검사를 더함
속도와 비용	몇 초 안에 답해야 하는가, 질문 하나에 얼마까지 쓸 수 있는가	실시간 대화는 한 번 검색하는 구조, 조사형 업무는 여러 번 찾는 구조도 가능
보안과 권한	사람마다 볼 수 있는 문서가 다른가, 외부 서비스로 보내도 되는가	권한 검사는 검색 단계에서 해야 함. 민감하면 자체 운용 모델도 검토

확인할 것	무엇을 보는가	구조에 미치는 영향
언어	한국어 전용인가, 한·영 혼합인가	한국어 형태소 분석(아래 설명)과 언어별 성능 확인이 필요

질문을 종류로 나눠 보는 표

질문 종류	예시	필요한 능력	맞는 첫 번째 방법
단일 사실 찾기	환불 기한이 며칠인가	정확한 조각 1~3개 찾기	하이브리드 검색
절차·조건 찾기	이 예외가 적용되는 다섯 가지 조건은	목록과 표가 온전히 유지될 것	구조를 살리는 나누기 + 검색
여러 문서 견주기	A와 B의 공급망 차이는	여러 출처를 동시에 찾고 비교	질문 확장 + 검색, 에이전트 방식
관계·다중 홉	X와 계약한 회사가 Y 소송과 어떻게 연결되나	대상 사이의 관계 추적	그래프 방식(GraphRAG)
전체 요약	1만 건 문의의 핵심 주제는	전체를 모아 주제 뽑기	그래프 전역 검색, 나눠 요약 후 합치기
최신·외부 정보	어제 바뀐 정책은	새로움, 실시간 자료원	검색 + 웹·업무 시스템 도구

가장 먼저 할 일은 기술 선택이 아닙니다

실제 사용자가 던질 **질문 50~100개와, 각 질문의 정답이 어느 문서 어디에 있는지**를 모아 두세요. 이것이 평가 집합(Evaluation Set)입니다. 이것이 있어야 조각 크기, 검색 방식, 재정렬 사용 여부를 감이 아니라 숫자로 비교할 수 있습니다. 쉬운 질문만 넣으면 점수가 거짓말을 하므로, 답이 자료에 없는 질문, 연결형 질문, 최신 정보 질문도 반드시 섞어야 합니다.

---

3. 자료 준비 — 답의 품질은 여기서 70%가 결정됩니다

3.1 모으기와 해석하기: 쓰레기가 들어가면 검색도 쓰레기가 됩니다

- PDF, 스캔 문서, 표, 발표 자료는 구조를 살려 읽어 내는 전용 도구(파서, Parser)를 씁니다. 표는 글로 뭉개지 말고, 열 제목과 행의 구조를 유지합니다.
- 웹 문서와 마크다운(Markdown, 제목 기호로 구조를 표시하는 간단한 문서 형식)은 제목의 상하 관계를 정보로 보존합니다. 이 제목 경로가 나중에 조각의 문맥을 되살리는 가장 값싼 재료가 됩니다.

- 중복 문서, 옛 판본, 초안은 지우거나 판본 정보를 붙입니다. 같은 정책의 옛 판본과 새 판본이 함께 들어가면 AI는 둘을 섞어 답합니다.
- 모든 조각에 최소한의 꼬리표(메타데이터, Metadata — 자료에 붙이는 설명 정보)를 붙입니다. 출처 문서, 위치(절·쪽), 만든 날짜와 고친 날짜, 소유자, 볼 수 있는 사람의 범위, 문서 종류가 필요합니다.

자료 종류별로 신경 쓸 점도 다릅니다.

원본 자료	따로 챙길 것
PDF, 워드 문서	문단·제목·표·쪽 구조 보존
파워포인트	발표 자료와 발표자 메모를 연결
엑셀, CSV 표 파일	표의 구조(열 이름, 자료 형태) 추출
이메일	보낸 사람, 받는 사람, 답장 흐름(스레드), 날짜
메신저·채팅	대화 흐름, 답글 관계, 시점
회의 녹취	말한 사람, 시간, 논의·결정·할 일 구분
관계형 데이터베이스	표 사이의 관계, 칸의 뜻, 권한
그림이 든 문서	표·도표·그림의 내용을 따로 해석

원본 파일, 뽑아낸 글, 표 구조, 원본에서의 위치를 서로 연결해 보관하는 것이 중요합니다. 이 연결이 끊기면 나중에 출처를 댈 수 없습니다.

3.2 나누기(청킹, Chunking): 정답은 없고, 자료 종류별 기본값이 있습니다

연구들이 공통으로 말하는 교훈은 두 가지입니다. 첫째, '똑똑해 보이는' 나누기 방법이 항상 이기지는 않습니다. 둘째, 조각이 너무 작으면 관련 조각을 찾아내는 비율은 올라가도, 정작 답을 만들 문맥이 부족해집니다. **많이 찾았는데 답은 틀리는** 함정입니다.

실무에서 검증이 많이 된 출발점은 **구조를 따라 재귀적으로 나누기(Recursive Splitting)**입니다. 문단, 그다음 문장, 그다음 단어 순으로 자연스러운 경계를 우선해 자르며, 크기는 약 512 토큰, 겹침(오버랩, Overlap — 이웃 조각과 일부를 겹치게 두는 것)은 10~20%가 기본값입니다.

자료 종류	권장하는 나누기	피해야 할 것
일반 글·보고서	문단 경계 우선, 최대 약 512~800 토큰	문단 중간을 임의로 자르기
마크다운·기술 문서	제목 단위로 나누고 제목 경로를 꼬리표로 보관	제목과 본문을 떼어 놓기
프로그램 코드	함수·클래스 단위 유지	함수 본문 중간 자르기
표	열 제목을 모든 행 묶음 조각에 반복해 넣기	제목 없는 행만 나누기

자료 종류	권장하는 나누기	피해야 할 것
목록	소개 문장을 각 항목 묶음에 반복해 넣기	항목만 떼어 나누기
법률·의료·표준 문서	계층 나누기 — 작은 조각에 상위 장·절 표시를 함께 넣기	조항 번호와 조건을 떼어 놓기
대화·회의록	말한 사람·주제가 바뀌는 지점으로 나누기	토큰 수만으로 자르기

나누는 방식 자체도 여러 가지가 있습니다. 일정 크기로 자르는 고정 크기(Fixed-size), 의미가 바뀌는 지점에서 자르는 의미 기반(Semantic), 제목·절·표 구조를 따르는 구조 인식(Structure-aware) 방식 등입니다. 가장 실용적인 조합은 구조 인식 나누기를 기본으로 하고, 아래 두 기법을 더하는 것입니다.

부모-자식 나누기(Parent-Child Retrieval) — 작은 자식 조각으로 정확히 찾되, 답을 만들 때는 그 조각이 속한 부모 절 전체를 AI에게 건넵니다. '찾는 단위'와 '읽는 단위'를 나누면 정확함과 문맥을 동시에 얻을 수 있습니다. 시작점으로는 검색용 조각 300~600 토큰, 부모 문맥 1,500~3,000 토큰을 비교해 볼 수 있지만, 보편적인 정답값은 아니므로 내 평가 집합으로 확인해야 합니다.

늦은 나누기 / 레이트 청킹(Late Chunking) — 순서를 뒤집는 방법입니다. 먼저 문서 전체를 읽어 글자 하나하나의 의미 표현을 만든 뒤, 그다음에 조각 경계로 나눕니다. 각 조각에 문서 전체의 문맥이 스며들어, 다른 곳에서 설명된 대상을 가리키는 조각도 잘 찾아집니다. 뒤에서 설명할 문맥 기반 검색과 달리 AI를 불러 설명문을 만들 필요가 없어 비용이 들지 않습니다. 대신 문서 길이가 임베딩 모델이 한 번에 읽을 수 있는 한도(예: 8,192 토큰, 약 10쪽)를 넘으면 쓸 수 없습니다.

3.3 저장하기(인덱싱, Indexing): 한 번 잘못 고르면 전부 다시 만듭니다

인덱싱은 '나중에 찾기 좋게 정리해 저장소에 넣는 일'입니다.

- **의미 좌표 모델(임베딩 모델, Embedding Model)을 바꾸면 전부 다시 만들어야 합니다.** 숫자 공간 자체가 달라지기 때문입니다. 모델 이름과 판본을 조각마다 기록하고, 바꿀 때는 새것과 옛것을 함께 쓰는 전환 계획을 세웁니다.
- **바뀌지 않은 조각은 건너뜁니다.** 조각 내용과 모델 판본으로 만든 지문(해시, Hash)을 열쇠로 쓰면, 안 바뀐 조각은 다시 계산하지 않아 비용과 시간을 아끼고, 중간에 멈춘 작업도 안전하게 이어 갈 수 있습니다.
- **의미 검색용과 키워드 검색용, 두 가지를 모두 만듭니다.** 나중에 검색 단계에서 보완할 수 없기 때문입니다.
- **권한과 구분 정보는 저장할 때 넣습니다.** 날짜, 부서, 접근 등급 같은 걸러내기용 정보는 실제 검색에서 쓸 수 있게 처음부터 넣어 둬야 합니다. 검색한 뒤에 걸러내는 방식은 보안과 정확함 모두에서 실패합니다.

3.4 한국어 자료에서 특히 주의할 점

한국어는 단어 뒤에 조사와 어미가 붙는 언어(교착어)입니다. '계약서를'과 '계약서'를 컴퓨터는 다른 단어로 볼 수 있습니다. 그래서 키워드 검색에는 말을 최소 단위로 쪼개는 형태소 분석(Morphological Analysis)이 필요합니다. 노리(Nori), 키위(Kiwi), 메캡(MeCab) 같은 분석 도구가 이 일을 합니다. 어떤 분석기를 고르느냐가 검색 정확도에 직접 영향을 줍니다.

또 임베딩 모델의 한국어 실력은 영어 시험 성적과 다를 수 있습니다. 반드시 내 자료로 직접 비교해야 합니다. 한·영이 섞인 자료라면 다음 네 가지를 모두 시험해 보세요.

- 한국어 질문 → 한국어 문서
- 영어 질문 → 영어 문서
- 한국어 질문 → 영어 문서
- 회사 줄임말, 사람 이름, 전문 용어, 제품 번호·계약 번호 같은 정확한 단어 찾기

---

4. 찾기 단계 — 문맥 기반 검색(Contextual Retrieval)이 보여 준 '쌓이는 개선'

4.1 핵심 아이디어: 조각이 스스로를 설명하게 하기

앤트로픽(Anthropic)이 2024년 9월 19일 발표한 문맥 기반 검색(Contextual Retrieval)은 나누기의 문맥 손실 문제를, 저장하는 시점에 정면으로 해결합니다. 방법은 단순합니다.

1. 문서를 조각으로 나눕니다.
2. 조각마다 작은 AI에게 원문 전체와 그 조각을 함께 주고, '이 조각이 문서 안에서 무엇을 설명하는지'를 50~100 토큰의 짧은 설명으로 쓰게 합니다.
3. 그 설명을 조각 앞에 붙인 글로 의미 검색용 저장소와 키워드 검색용 저장소를 모두 만듭니다.

예를 들어 원래 조각이 '매출이 전 분기 대비 3% 늘었다'뿐이라면, 앞에 이런 설명이 붙습니다. '이 내용은 ACME 사의 2023년 2분기 실적 공시의 일부이며, 전 분기 매출은 3억 1,400만 달러였다.' 이제 이 조각은 스스로를 설명하므로, 'ACME 2분기 매출'이라는 질문에도 걸립니다.

이 기법은 발표된 지 2년이 지나 '최신 기술'이라기보다, 검증된 표준에 가깝습니다. 지금 검색 시스템을 운영한다면 가장 먼저 적용을 검토할 만한 방법입니다.

참고할 점이 네 가지 있습니다.

- **비용** — 문서 전체를 한 번 올려 두고(프롬프트 캐싱, Prompt Caching — 자주 쓰는 앞부분을 저장해 뒀다 재사용해 비용과 시간을 줄이는 기술) 조각마다 재사용하면, 문맥 설명을 만드는 비용은 문서 100만 토큰당 약 1.02달러로 추산됐습니다. 질문할 때마다가 아니라 저장할 때 한 번만 내는 비용입니다. (800 토큰 조각, 8,000 토큰 문서 가정, 발표 당시 클로드 3 하이쿠(Claude 3 Haiku) 기준)
- **문서 요약 전체를 붙이는 것과는 다릅니다.** 앤트로픽은 문서 요약 한 덩어리를 모든 조각에 붙이는 방식은 효과가 제한적이었다고 밝혔습니다. 조각마다 맞춤 설명을 다는 것이 핵심입니다.
- **용어집을 넣어 주면 더 좋아집니다.** 다른 문서에서만 뜻이 정해지는 용어가 있다면, 그 용어집을 설명 생성 지시에 넣어 주세요.
- **생성된 설명은 파생 자료입니다.** AI가 쓴 설명이 원문의 사실을 제대로 반영했는지 확인해야 합니다. 원문과 설명은 따로 보관하고, 답변의 출처 표시는 반드시 원문으로 연결하세요.

4.2 검증된 수치 — 각 단계는 서로 다른 실패를 잡습니다

앤트로픽이 여러 분야의 자료로 측정한 수치입니다. '검색 실패율'은 정답 조각이 상위 20개 안에 아예 없는 비율을 말합니다.

구성	상위 20개 검색 실패율	기준 대비 감소
기본 의미 검색만	5.7%	—
+ 문맥 임베딩(Contextual Embeddings)	3.7%	35%
+ 문맥 키워드 검색(Contextual BM25)	2.9%	49%
+ 재정렬(Reranking)	1.9%	67%

읽는 법이 중요합니다. 어느 한 기술이 혼자 다 해결하지 않습니다. 문맥 붙이기는 한 종류의 실패를, 키워드 검색 추가는 또 다른 종류를, 재정렬은 또 다른 종류를 구합니다. **검색 품질은 하나의 비법이 아니라, 서로 다른 실패를 하나씩 막는 층을 쌓아서 만듭니다.** 이것이 이 실험이 남긴 가장 실무적인 교훈입니다.

단, 이 수치는 앤트로픽의 시험 자료 기준입니다. 또한 '검색 실패가 67% 줄었다'는 뜻이지 '답변 정확도가 67% 올랐다'는 뜻이 아닙니다. 내 자료에서의 효과는 내 평가 집합으로 다시 재야 합니다.

4.3 두 검색 결과를 합치는 법

의미 검색 결과와 키워드 검색 결과는 점수 체계가 달라 그대로 더하면 한쪽이 지배합니다. 그래서 점수가 아니라 순위를 기준으로 합치는 상호 순위 결합(Reciprocal Rank Fusion, RRF)을 표준으로 씁니다. 각 검색에서 몇 번째로 나왔는지만 보고 합산하므로, 점수 크기의 차이에 휘둘리지 않습니다.

실무 기본 흐름은 이렇습니다. 넓게 약 150개를 뽑아 관련 있는 것을 놓치지 않고(재현율, Recall — 찾아야 할 것 중 실제로 찾은 비율), 재정렬로 20개로 좁혀 정확도를 높입니다(정밀도, Precision — 가져온 것 중 실제로 관련 있는 비율).

4.4 재정렬(Reranking): 싸게 넓게 찾고, 비싸게 좁게 고르기

재정렬 도구(리랭커, Reranker)는 질문과 조각을 쌍으로 함께 읽고 관련도를 다시 매깁니다. 미리 계산해 둔 숫자끼리 비교하는 것보다 비싸지만 훨씬 정확합니다. 그래서 역할이 나뉩니다. **검색은 싸게 넓게, 재정렬은 비싸게 좁게.** 몇 개를 재정렬할지는 답이 늦어지는 정도와 비용 사이의 맞교환입니다. 앤트로픽 실험에서는 최종적으로 상위 20개를 AI에게 건네는 구성이 5개나 10개보다 효과적이었지만, 너무 많으면 AI가 산만해질 수 있어 직접 평가해야 합니다.

4.5 질문 쪽을 고치는 기술들

검색 실패의 절반은 문서가 아니라 질문 쪽 문제입니다. 아래 기술은 저장소를 건드리지 않고 질문 단계에서 적용할 수 있어 도입이 빠릅니다.

- **질문 다시 쓰기(Rewrite)** — 앞 대화에 기대는 질문('그건 얼마야?')을, 혼자서도 뜻이 통하는 검색 질문으로 다시 씁니다.

- **질문 여러 개로 늘리기(Multi-query / RAG-Fusion)** — 질문을 여러 표현으로 바꿔 각각 검색하고 결과를 합칩니다. 여러 면을 가진 질문에 강합니다.
- **가상 답안으로 찾기(Hypothetical Document Embeddings, HyDE)** — AI가 먼저 모범 답안 같은 글을 쓰고, 그 글을 기준으로 검색합니다. 답안은 문서와 문체가 비슷해, 질문보다 목표 문서에 가까워지는 경우가 많습니다.
- **질문 쪼개기(Decomposition)** — 'A와 B를 비교하라'를 A 찾기와 B 찾기로 나눕니다.
- **조건으로 먼저 좁히기(Metadata Filtering / Self-Query)** — 질문에서 날짜, 부서, 문서 종류 같은 조건을 뽑아 검색 범위부터 좁힙니다.

---

5. 답 쓰기 단계 — 찾은 증거를 답으로 바꾸는 규칙

검색이 좋아도 답을 쓰는 지시가 허술하면 지어낸 답이 돌아옵니다. 최소 규칙은 다음과 같습니다.

- **근거만으로 답하게 하기** — '제공된 자료만으로 답하고, 없으면 모른다고 답하라'고 명시합니다. 단, 이 지시만으로 환각이 사라지지는 않으므로 아래 출처 검증과 짝을 이뤄야 합니다.
- **출처 붙이기(Citation)** — 주장마다 출처 조각을 표시하게 하고, 가능하면 답이 만들어진 뒤 그 출처가 실제로 그 주장을 뒷받침하는지 자동으로 검사합니다. 겉보기에만 출처가 있고 내용이 안 맞는 경우를 잡기 위해서입니다.
- **증거 꾸러미(Evidence Package)로 건네기** — 무조건 많은 문서를 넘기지 말고, 검증할 수 있는 원문, 문서 이름·판본·쪽 같은 정확한 위치, 적용되는 시간 범위, 발견된 관계, 문서끼리 어긋나는 점이나 빈 곳까지 묶어 건넵니다.
- **배치와 정리** — 가장 관련 높은 조각을 앞과 뒤에 배치하고(중간에 묻힌 정보는 놓치기 쉽습니다), 중복 조각은 지웁니다. 같은 문서의 조각은 원래 순서대로 정렬하면 이야기의 흐름이 살아납니다.
- **압축해서 건네기** — 조각 전체 대신 질문에 답하는 문장만 뽑아 건네면 토큰 비용을 30~60% 줄이면서 방해 요소도 줄일 수 있습니다. 다만 압축 과정 자체가 정보를 잃을 수 있어 평가로 확인해야 합니다.
- **못 찾으면 못 찾았다고 답하기** — 관련도가 기준 아래면 추측성 답변 대신 '자료에서 찾지 못했다'고 답하고, 어떤 자료를 찾았는지 보여 주는 것이 신뢰를 지킵니다. AI가 스스로 매긴 확신 점수를 정확도의 증거로 삼아서는 안 됩니다. 출처가 충분한지를 기준으로 상태를 표시하는 편이 낫습니다.
- **사실과 추론 구분하기** — 원문에 있는 사실과, 그로부터 이끌어 낸 추론을 답 안에서 구분해 표시하게 합니다.

---

6. 관계를 다루는 방법 — 그래프(Graph)와 계층 요약

6.1 그래프 검색 증강 생성(GraphRAG)이란

마이크로소프트 리서치(Microsoft Research)가 2024년 제안한 방법입니다. 저장하는 단계에서 AI가 문서 전체에서 대상(개체, Entity — 인물·조직·개념 등)과 관계(제공한다, 소속된다, 썼다 등)를 뽑아 지식 그래프(Knowledge Graph)를 만들고, 서로 가깝게 연결된 대상 묶음(커뮤니티, Community)마다 요약을 미리 만들어 둡니다.

질문에 답하는 방식은 네 가지입니다.

- **지역 검색(Local Search)** — 특정 대상 주변을 살펴 세부 질문에 답합니다. 예: 'A 프로젝트의 책임자는 누구인가'
- **전역 검색(Global Search)** — 커뮤니티 요약들을 나눠 읽고 합치는 방식(맵-리듀스, Map-Reduce)으로 전체 자료 질문에 답합니다. 예: '전체 프로젝트의 공통 문제는 무엇인가'
- **드리프트 검색(DRIFT Search)** — 전체 맥락에서 시작해 점점 구체적인 후속 검색으로 좁혀 갑니다.
- **기본 검색(Basic Search)** — 일반 조각 검색으로 대신합니다.

강점은 분명합니다. 여러 단계를 연결하는 다중 홉(Multi-hop) 질문과 전체 요약에서 평면적인 조각 검색과 격차가 벌어지고, 그래프의 경로를 따라가므로 답의 근거를 구조적으로 추적할 수 있습니다.

대가도 분명합니다. 그래프를 만드는 비용이 기본 방식의 3~5배 수준으로 들고, 같은 대상의 다른 표기('Microsoft'와 'Microsoft Corporation', '삼성전자'와 'Samsung')를 하나의 점으로 합치는 일 — 개체 중복 해소(Entity Resolution) — 의 품질이 전체 품질을 좌우합니다. 단순한 사실 찾기가 대부분인 시스템에 도입하면 비용만 늡니다.

6.2 같은 이름, 전혀 다른 비용 — 그래프 변종들

'GraphRAG'라는 이름 아래에는 비용 구조가 완전히 다른 변종이 여럿 있습니다. 도입을 검토할 때는 어떤 변종인지부터 확인해야 합니다.

변종	핵심 아이디어	강점	약점·비용
마이크로소프트 그래프 RAG(Microsoft GraphRAG)	대상·관계를 뽑아 그래프를 만들고 커뮤니티별 요약을 미리 생성	전역 질문에서 답의 포괄성과 다양성 향상	저장할 때 조각마다 AI 호출. 자료가 바뀌면 다시 만드는 부담. 질문당 건네는 글이 매우 길어짐
레이지그래프 RAG(LazyGraphRAG)	미리 요약 만들기를 생략하고 그래프 처리를 질문 시점으로 미룸	저장 비용이 의미 검색 방식과 같고 기존 그래프 RAG의 0.1% 수준. 전역 질문 비용을 700배 이상 줄이면서 품질은 비슷	질문 시점의 계산이 늘어남. 관련성 검사 예산으로 비용과 품질을 조절해야 함
라이트RAG(LightRAG)	그래프 저장 + 구체 수준과 주제 수준의 이중 검색	새 자료를 덧붙이기 쉬워 갱신에 강함. 오픈소스	질문당 건네는 글이 약 1만 토큰으로 큰 편
히포RAG(HippoRAG / HippoRAG 2)	정해진 틀 없이 그래프를 만들고, 연상 기억처럼 관련	다중 홉 질문에 강하고 건네는 글이 짧음(약 1천 토큰). 자료가 커져도 성능이 안정적	대상 연결이 잘못되면 엉뚱한 방향으로 퍼짐

변종	핵심 아이디어	강점	약점·비용
	도를 퍼뜨려 한 번에 다중 홉을 탐색		
랩터(RAPTOR)	조각을 묶어 요약하기를 반복해 아래에서 위로 요약 나무(트리)를 만듦	세부와 개요를 여러 추상 수준에서 검색 가능	요약 나무를 다시 만드는 비용, 요약 과정의 정보 손실

6.3 그래프가 이기는 곳과 지는 곳 — 균형 잡힌 연구 결과

국제 학회 ICLR 2026에 발표된 그래프RAG 벤치(GraphRAG-Bench) 연구가 가장 균형 잡힌 판단 근거를 줍니다. 핵심 관찰은 세 가지입니다.

- **단순한 사실 찾기는 기본 방식이 같거나 더 낫습니다.** 그래프가 관련은 있지만 필요 없는 정보까지 끌어와 답의 품질을 떨어뜨립니다. 이 연구가 인용한 앞선 연구에서는 그래프 방식이 일반 질의응답에서 기본 방식보다 정확도가 13.4% 낮았습니다.
- **복잡한 추론과 맥락 요약에서는 그래프가 확실히 앞섭니다.** 멀리 떨어진 문서 조각들을 연결해야 하는 질문일수록 차이가 커집니다.
- **건네는 글의 팽창을 관리해야 합니다.** 기본 방식의 질문당 문맥이 약 900 토큰일 때, 마이크로소프트 그래프RAG의 전역 검색은 수만 토큰까지 커졌습니다. 많이 가져오는 것이 곧 잡음입니다.

흥미로운 결과도 있습니다. 자료를 약 5만 6천 토큰에서 113만 토큰으로 키우자 기본 방식의 복잡 추론 정확도는 58.6%에서 43.2%로 떨어졌지만, 히포RAG 2(HippoRAG 2)는 거의 그대로였습니다. 자료가 커질수록 '비슷하지만 틀린' 조각이 늘어나는 의미 검색의 약점을, 그래프의 명시적인 대상 일치가 걸러 준다는 해석입니다.

6.4 그래프에 대한 실무 조언

- **이미 구조가 있는 자료는 AI로 뽑지 말고 직접 넣으세요.** 조직도, 고객-계약-담당자 관계, 프로젝트 이력처럼 틀이 명확한 자료는 그래프 데이터베이스에 직접 넣고 그래프 질의(예: 텍스트-투-사이퍼, Text-to-Cypher — 말로 된 질문을 그래프 질의 언어로 바꾸는 방식)로 답하는 편이 정확하고 저렴합니다. AI로 그래프를 뽑는 도구는, 구조 없는 글에서 관계를 '발견'해야 할 때 쓰는 것입니다.
- **관계에도 출처와 유효 기간이 필요합니다.** 관계가 어느 원문에서 왔는지, 언제부터 언제까지 성립하는지를 함께 저장해야 '과거에 담당했던 프로젝트를 지금도 담당한다'고 답하는 일을 막을 수 있습니다.
- **그래프는 기본 방식을 대체하기보다 보완합니다.** 한 곳에서 찾는 질문은 하이브리드 검색으로, 연결·전역 질문만 그래프 경로로 보내는 길 나누기(라우팅, Routing)가 현실적입니다.
- **도입 전에 유지 상태를 확인하세요.** 마이크로소프트의 공개 그래프RAG 연구 저장소는 현재 유지보수 중심 상태라고 스스로 밝히고 있습니다. 연구 구조를 참고하는 것과, 그 구현 자체를 장기 제품의 핵심 의존성으로 삼는 것은 구분해야 합니다.

계층 요약 방식인 랩터(RAPTOR)도 같은 맥락에서 이해하면 됩니다. 세부 사실부터 상위 개념까지 여러 수준으로 요약 나무를 만들어 두면, '전체 주제' 같은 개념적 질문에도 답할 수 있습니다. 긴 문서 질의응답 시험에서 정확도가 크게 향상된 보고가 있지만, 요약 과정에서 정보가 사라지는 대가는 평가로 확인해야 합니다.

7. 미리 정리해 두는 방법 — 엘엘엠 위키(LLM Wiki)

7.1 발상을 뒤집기: 찾을 때 정리하지 말고, 넣을 때 정리하기

엘엘엠 위키(LLM Wiki)는 안드레이 카르파티(Andrej Karpathy)가 2026년 4월 제안한 패턴으로, 기본 검색 방식과 방향이 정반대입니다. 질문이 올 때마다 원본 조각을 뒤지는 대신, **자료를 넣는 시점에 AI가 원본을 읽고 사람이 읽을 수 있는 마크다운 위키로 미리 합성(컴파일, Compile)해 둡니다.**

컴파일러에 비유하면 이해가 쉽습니다. 프로그래밍에서 컴파일러는 사람이 쓴 원본 코드를 기계가 실행할 수 있는 형태로 미리 번역해 두는 도구입니다. 여기서 원본 자료는 소스 코드, AI는 컴파일러, 위키는 번역 결과물, 정기 점검은 시험, 질문에 답하는 일은 실행에 해당합니다.

7.2 세 층과 세 가지 작업

구성 요소	역할	핵심 규칙
원본 층 `raw/`	논문·기사·메모·PDF 등 모은 자료	AI는 읽기만 하고 절대 고치지 않음. 모든 주장의 근거
위키 층 `wiki/`	AI가 쓰고 유지하는 마크다운 문서. 출처 요약, 개념·대상별 페이지, 비교와 종합	모든 페이지에 원본 출처 연결 필수
목차 `wiki/index.md`	전체 페이지 목록과 한 줄 요약	질문이 오면 AI가 가장 먼저 읽는 지도
기록 `log.md`	수집·질의·점검 기록	지우지 않고 덧붙이기만 함. 나중에 추적하는 근거
규칙 파일(Schema)	폴더 구조, 페이지 양식, 이름 규칙, 작업 절차를 적은 파일. 예: `CLAUDE.md`, `AGENTS.md`	가장 중요한 파일. 모호하면 위키도 엉망이 됨

세 가지 작업이 순환합니다.

1. **수집(Ingest)** — 새 자료를 원본 층에 넣으면 AI가 읽고 관련 위키 페이지 10~15개를 고치거나 새로 만들고, 문서 사이의 연결(백링크)과 목차를 갱신합니다.
2. **질의(Query)** — AI가 목차를 먼저 읽고 필요한 페이지만 열어 출처와 함께 답합니다. 좋은 답변은 다시 위키 페이지로 저장돼, 질문할수록 지식이 쌓입니다.
3. **점검(Lint)** — 페이지끼리 어긋나는 내용, 낡은 주장, 아무도 연결하지 않는 페이지, 자주 언급되는데 페이지가 없는 개념을 주기적으로 찾아냅니다. 건강 검진과 같습니다.

카르파티의 사례는 약 100개 문서, 40만 단어 규모에서 목차가 AI의 문맥 안에 충분히 들어가 벡터 저장소 없이 동작했습니다.

7.3 기본 검색 방식과의 본질적인 차이

핵심 차이는 **정리 비용을 언제 치르느냐**입니다. 기본 방식은 질문할 때마다 조각을 찾아 매번 다시 종합합니다(필요할 때 찾기, Just-in-time). 위키는 넣을 때 한 번 깊게 읽고, 서로 참조하게 하고, 어긋나는 내용까지 정리해 둡니다(미리 정리하기, Compile-ahead). 그래서 다섯 문서를 종합해야 하는 미묘한 질문에 강하고, 나누기 문제 자체가 사라집니다. 앞에서 본 문맥 기반 검색이 '조각마다 문맥을 미리 써 붙이는' 것과 같은 철학을, 문서 전체 수준으로 밀어붙인 셈입니다.

원본이 자주 바뀌면 필요할 때 찾기가, 지식이 쌓이고 재사용될수록 미리 정리하기가 유리합니다.

7.4 한계와 위험 — 두 층을 나눠야 안전합니다

- **규모의 상한** — 실무 보고로는 단일 주제 100~400개 문서, 약 40만 단어 수준이 한계입니다. 그 이상이면 목차만으로 탐색하기 어려워져 검색 도구가 다시 필요합니다. 카르파티도 마크다운용 지역 하이브리드 검색 도구인 큐엠디(qmd — 키워드 검색 + 의미 검색 + AI 재정렬)를 확장 수단으로 언급했습니다.
- **오류의 누적** — AI가 잘못 연결한 관계가 위키에 남으면 이후 모든 답변에 영향을 줍니다. 반복해서 다시 쓰는 과정에서 미묘한 오류가 쌓일 위험도 있습니다. 원본 불변 원칙, 출처 연결, 판본 기록(Git 이력), 정기 점검이 안전장치입니다.
- **사람 글과의 분리** — AI가 만든 내용은 개인 노트와 분리된 공간에 두는 것이 좋습니다. 무엇이 내가 쓴 글이고 무엇이 AI가 정리한 글인지 섞이면 나중에 믿을 근거가 흐려집니다.
- **조직 운영 요건 부재** — 역할별 권한, 동시 편집, 감사 기록이 없어 기업 전체 지식 저장소로는 맞지 않습니다.

그래서 두 층을 구분하는 원칙이 중요합니다.

층	담는 것	믿는 정도
증거 층(Evidence Layer)	실제 원본 문서와 기록	사실 확인의 기본 근거
지식 층(Knowledge Layer)	위키, 그래프, 요약, 임베딩	찾기·탐색·추론을 돕는 파생 지식

위키와 그래프는 유용한 '미리 만들어 둔 지식 보기(Materialized Knowledge View)'이지만, 원본의 권위를 자동으로 물려받지는 않습니다. AI가 쓴 위키 내용을 다시 AI가 아무 검증 없이 사실로 쓰는 순환 구조는 특히 위험합니다.

현실적인 조합은, **자주 쓰고 계속 바뀌는 '뜨거운' 지식은 위키로, 거의 안 바뀌는 대량 보관 자료는 검색 방식으로** 나누는 것입니다. 자료가 이미 마크다운이면 둘 사이를 오가는 비용도 거의 없습니다.

---

8. 검색 자체를 바꾸는 접근 — 길게 넣기, 캐시, 에이전트, 직접 연결

세 번째 흐름은 '미리 쪼개서 저장해 둔다'는 전제 자체를 버리는 것입니다. 자료가 작으면 통째로 넣고, 크면 에이전트가 도구로 필요한 만큼 찾아 읽게 합니다. 다만 어느 쪽이든 'AI에게 무엇을 보여 줄지 고르는 문제'는 사라지지 않습니다.

8.1 긴 문맥(Long Context)과 캐시 증강 생성(CAG): 작으면 그냥 다 넣기

앤트로픽은 문맥 기반 검색 글에서, 지식 자료가 20만 토큰(약 500쪽) 미만이면 검색 시스템을 만들지 말고 전체를 프롬프트에 넣으라고 권했습니다. 프롬프트 캐싱(Prompt Caching)을 쓰면 반복 질문의 대기 시간은 절반 이하로, 비용은 최대 90%까지 줄어듭니다.

학계에서는 이를 캐시 증강 생성(Cache-Augmented Generation, CAG)으로 정리했습니다. 관련 문서를 모두 AI의 확장 문맥에 미리 올려 두고 내부 계산 결과(캐시)를 미리 만들어 두면, 검색 지연과 검색 오류가 사라지고 시스템도 단순해집니다. 지식 자료의 크기가 제한적일 때 기본 검색 방식과 같거나 더 나은 결과를 보였습니다. 상용 AI 서비스를 인터페이스(API)로 쓰는 경우에는 프롬프트 캐싱이 사실상 같은 역할을 합니다. 단점은 자료가 바뀔 때마다 캐시를 다시 만들어야 하고, 첫 요청 비용이 크다는 점입니다.

8.2 하지만 길수록 흐려집니다 — 문맥 부패(Context Rot)

크로마(Chroma)가 2025년 7월 발표한 문맥 부패(Context Rot) 연구는, 18개 최신 모델 모두가 입력이 길어질수록 성능이 떨어진다는 것을 보였습니다. 숨은 바늘 찾기 같은 단순 시험에서는 거의 만점이지만, 비슷하지만 틀린 방해 정보가 섞이거나 의미를 이해해야 하면 길이에 따라 정확도가 내려갑니다.

비슷한 현상은 예전부터 알려졌습니다. 긴 문맥의 중간에 묻힌 정보는 놓치기 쉽다는 중간 소실(Lost in the Middle) 현상입니다. 앤트로픽도 이를 AI의 '주의력 예산'으로 설명하며, 문맥을 한정된 자원으로 다루라고 합니다.

결론은 같습니다. **넣을 수 있다는 것과 잘 쓴다는 것은 다릅니다.** 계약서 한 건을 통째로 분석하는 단일 작업에는 긴 문맥이 최적이지만, 대규모 지식 자료의 상시 대체재는 아닙니다. 활용할 때는 문서 순서와 질문 위치, 불필요한 내용 제거, 문서별 출처와 판본 유지를 챙기고, 긴 입력에서의 성능을 평가 집합으로 확인해야 합니다.

8.3 에이전트형 검색(Agentic Retrieval): 저장해 두지 않고, 그때그때 찾아 읽기

코딩 도구 클로드 코드(Claude Code)를 만든 보리스 체르니(Boris Cherny)는, 초기 판본이 검색 방식과 지역 벡터 저장소를 썼지만 에이전트형 검색이 대체로 더 잘 작동하고 더 단순하며 보안·개인정보·최신성 문제도 적어 바꿨다고 밝혔습니다. 이 도구는 핵심 문서는 처음부터 문맥에 넣고, 나머지는 파일 이름 패턴 찾기, 내용 검색(그렙, grep — 파일에서 특정 글자가 든 줄을 찾는 전통 도구), 파일 읽기로 필요할 때 찾아오는 혼합 방식을 씁니다.

앤트로픽은 이를 필요할 때 가져오기(Just-in-time) 문맥이라고 부릅니다. 자료를 미리 다 처리하지 않고 파일 경로, 저장된 질의, 연결 주소 같은 가벼운 참조만 들고 있다가, 도구로 필요한 부분만 불러옵니다. 폴더 구조, 파일 이름, 수정 시각 같은 꼬리표 자체가 '어디를 봐야 할지'의 신호가 됩니다.

대가도 분명합니다.

- **느립니다.** 실행 중에 탐색하는 일은 미리 계산된 검색보다 느립니다.
- **도구와 안내 설계가 필요합니다.** 안내가 부실하면 에이전트가 막다른 길을 헤매며 문맥을 낭비합니다.
- **같은 탐색을 매번 반복합니다.** 임베딩은 일종의 '미리 계산해 둔 캐시'로 볼 수 있습니다. 한 발표의 예시에서는 매번 내용 검색으로 코드를 찾아 읽는 데 약 6,300 토큰이 든 반면, 저장된 조각 검색은 약 420 토큰이었습니다. 단, 이는 예시용 기록이며 통제된 시험 결과는 아닙니다.

그래서 업계가 모이는 지점은 '검색 방식 대 에이전트'가 아니라, **에이전트가 키워드 검색·의미 검색·내용 검색·데이터베이스 질의 같은 여러 찾기 도구를 골라 쓰며 반복 탐색하는 구조**입니다.

관련된 개념을 함께 정리해 둡니다.

- **자기 점검형(Self-RAG)** — AI가 검색이 필요한지 스스로 판단하고, 답을 쓰면서 근거를 스스로 점검합니다.
- **교정형(Corrective RAG, CRAG)** — 찾은 결과를 평가해 부실하면 질문을 바꿔 다시 찾거나 웹 검색으로 대체합니다.
- **에이전트형(Agentic RAG)** — 질문을 쪼개고, 자료원마다 다른 저장소를 골라 여러 차례 찾고 생각하기를 반복하며, 충분하다고 판단될 때 멈춥니다.

공통의 대가는 호출이 늘어 답이 수 초 늦어지고 비용이 커진다는 점, 그리고 멈추는 조건을 설계하지 않으면 무한 반복에 빠진다는 점입니다. 단순 조회에까지 에이전트를 씌우는 것은 과한 설계입니다. 쉬운 질문은 고정 흐름으로, 어려운 질문만 에이전트 경로로 보내는 적응형 길 나누기(Adaptive Routing)가 균형점입니다.

8.4 직접 연결하기 — 모델 컨텍스트 프로토콜(MCP)

모델 컨텍스트 프로토콜(Model Context Protocol, MCP)은 AI가 외부 시스템과 연결하는 공통 규약입니다. 구글 드라이브, 고객 관리 시스템, 사내 데이터베이스, 노션 같은 원본 시스템을 에이전트의 도구로 노출하면, 자료를 별도 저장소로 복사하지 않고도 원본을 직접 조회할 수 있습니다.

가장 큰 장점은 늘 최신이라는 점과, 원본 시스템의 권한 체계를 그대로 쓸 수 있다는 점입니다. 반대로 원본 시스템의 검색 품질과 호출 속도 제한에 그대로 묶입니다. 자주 묻는 핵심 자료는 별도 저장소나 위키로 정리해 두는 혼합 구성이 현실적입니다.

---

9. 자료 모양별로 따로 봐야 할 도구들

'내 데이터'가 줄글만은 아닙니다. 표·숫자, 발표 자료·스캔 문서, 그리고 '말투·형식'처럼 검색으로 풀 수 없는 영역은 각기 다른 도구가 맞습니다.

자료·요구	맞는 접근	왜	주의점
발표 자료, 스캔 PDF, 표·도표가 많은 보고서	페이지 그림 기반 검색 (ColPali 계열)	글자 추출·문자 인식·지면 복원 단계에서 정보가 사라지는 문제를 건너뜀	쪽마다 벡터 수가 많아 저장 공간이 큼
매출, 건수, 기간별 합계 등 숫자 질문	텍스트-투-SQL(Text-to-SQL)	검색은 '세기'와 '합하기'를 못 함. 정답은 데이터베이스에 있음	읽기 전용 계정, 지표 정의 층, 만든 질의 검증 단계 필수
사람·조직·계약의 관계	직접 모델링한 그래프 데이터베이스 + 그래프 질의	관계가 이미 명확하면 뽑아내는 것보다 정확	구조 설계와 동기화 비용

자료·요구	맞는 접근	왜	주의점
문체, 답변 형식, 분류 기준	미세 조정(Fine-tuning) 또는 예시 중심 지시	행동·형식은 학습이 효과적	사실 주입용으로는 부적합
사용자별 선호, 진행 중인 작업 상태	에이전트 메모리(Agent Memory) — 파일 기반 노트	대화를 넘어 상태를 유지	무엇을 저장하고 언제 지울지 규칙 필요

9.1 페이지 그림 기반 검색 — 콜팔리(ColPali)

발표 자료나 스캔 문서에서 가장 큰 손실은, 그림을 글자로 옮기는 단계에서 일어납니다. 페이지 그림 기반 검색(Visual Document Retrieval)은 그 단계를 건너뜁니다. 문서 쪽 전체를 그림으로 보고, 그림 자체의 의미로 찾는 방식입니다. 대표 연구가 콜팔리(ColPali, 2024)이며, 시각 문서 검색 시험에서 기존 글자 기반 흐름을 크게 앞섰습니다(ICLR 2025).

대가는 저장 공간입니다. 쪽마다 벡터가 많이 생깁니다. 문서가 적으면, 그림을 보고 이해할 수 있는 AI(멀티모달 모델, Multimodal Model)에게 쪽 그림을 직접 건네는 편이 더 단순할 수 있습니다.

9.2 숫자는 검색으로 흉내 내지 않기 — 데이터베이스 질의

가격, 재고, 일정, 수치 합계처럼 이미 표 형태로 있는 자료는, 조각으로 쪼개 의미 좌표를 붙이는 순간 정확도를 잃습니다. '지난달 지역별 매출 합계'는 의미 검색이 아니라 데이터베이스 질의 언어(SQL)로 답해야 합니다.

주의점은 세 가지입니다. 읽기만 가능한 계정을 쓸 것, '매출' 같은 지표의 정의를 미리 정리한 층(시맨틱 레이어, Semantic Layer)을 둘 것, AI가 만든 질의를 실행 전에 검증할 것. 문서형 지식(검색)과 구조화 조회(도구)를 질문마다 나눠 맡기는 구조가 기업 시스템의 표준 형태입니다.

9.3 미세 조정(Fine-tuning) — 사실이 아니라 행동을 가르치기

역할 구분을 한 문장으로 정리하면 이렇습니다. **검색은 지식을 주고, 미세 조정은 행동을 다듬으며, 긴 문맥은 작업 기억입니다.**

- 미세 조정이 이기는 경우 — 출력 형식(정해진 양식 지키기), 일관된 말투, 분류 기준, 분야 특유의 판단 습관처럼 '사실이 아니라 방식'의 문제. 수천 건의 정답 표시가 된 예시가 있을 때 효과가 검증됩니다.
- 미세 조정이 지는 경우 — 매주 바뀌는 사실을 외우게 하기. 고칠 수 없고, 출처를 댈 수 없고, 사람별 권한을 적용할 수 없는 '낡은 지식의 동결'이 됩니다.
- 실무 조합은 대개 검색(사실) + 시스템 지시(정책) + 필요할 때 가벼운 경량 미세 조정(로라, LoRA — 모델의 일부만 효율적으로 훈련하는 기법)입니다.

참고로, 잘 정리·점검된 위키를 학습 자료로 삼아 작은 모델을 미세 조정하는 방향도 향후 아이디어로 거론됩니다. 다만 이는 정리된 지식 층이 먼저 있어야 가능한 다음 단계입니다.

9.4 자료를 어떻게 나눠 담을 것인가 — 데이터 모델(Data Model)

본격적으로 만들 때는 최소한 다음 개념들을 구분해 보관하는 것이 좋습니다. 이름은 기술 용어 그대로 두되, 뜻은 우리말로 풀었습니다.

담는 것	꼭 필요한 정보
문서(Document)	문서 번호, 원본 주소, 판본, 내용 지문(체크섬), 소유자
절(Section)	절 번호, 속한 문서, 제목 경로, 쪽 범위
조각(Chunk)	조각 번호, 속한 절, 원문, 문맥 설명, 임베딩
사실·주장(Fact / Claim)	주장 번호, 주어, 관계, 목적어, 원문 위치
대상(Entity)	대상 번호, 대표 이름, 다른 표기들, 종류
관계(Relation)	출발 대상, 도착 대상, 관계 종류, 근거 번호
위키 페이지(Wiki Page)	주제 번호, 요약, 근거 주장 번호들, 검토 상태

여기에 공통으로 붙는 정보도 중요합니다. 조직 구분(테넌트, Tenant), 접근 정책, 만든 시각, 유효 시작·종료 시점, 사용한 모델 판본, 뽑아낸 방식의 판본 등입니다. 특히 그래프에서는 관계의 출처와 유효 시점이 중요합니다.

---

10. 권장 구조 — 혼합 지식 구조(Hybrid Knowledge Architecture)

대부분의 실용적인 시스템에는 처음부터 전면적인 그래프 방식을 도입하기보다, 단계적으로 확장하는 설계를 권합니다. 핵심은 저장소를 여러 개 쓰는 것이 아니라, 원본·검색용 정리·관계 지식·종합 지식을 서로 분리해 관리하는 것입니다.

10.1 네 개 층으로 나누기

층	담는 것	쉬운 비유
1층 원본 증거(Raw Evidence)	원본 문서·데이터를 바꾸지 않고 보존	원본 서류 보관함
2층 검색 가능 지식(Searchable Knowledge)	조각, 임베딩, 키워드 색인, 꼬리표	도서관의 검색 카드
3층 구조화 지식(Structured Knowledge)	대상, 관계, 주장, 사건, 시간 정보	인물 관계도
4층 종합 지식(Synthesized Knowledge)	위키, 주제별 요약, 비교 보고서, 분석 결과	정리된 조사 보고서

장기적으로 보면 이것은 단순한 검색 시스템이 아니라, 개인이나 조직의 경험과 판단 근거를 재사용 가능한 지식 자산으로 바꾸는 시스템 — 지식 운영체제(Knowledge OS) — 가 됩니다. 예를 들어 컨설팅 자료가 쌓이는 경우,

두 번째 프로젝트에서 예전 원문을 새로 뒤지는 대신 이미 구축된 위키와 그래프로 관계를 탐색하고 새 정보만 갱신할 수 있습니다.

대가도 있습니다. 위키 내용이 바뀌면 관련 색인, 그래프, 요약, 캐시가 함께 갱신돼야 합니다. 단순 검색 방식에서는 덜 드러나는 운영상의 어려움입니다.

10.2 질문 길잡이(Query Router) — 질문마다 다른 길로 보내기

질문 예시	보낼 길	이유
특정 계약의 해지 조건은?	하이브리드 검색 + 부모-자식	조항·예외·원문을 함께 확보해야 함
모든 프로젝트의 공통 실패 원인은?	그래프 전역 검색 또는 위키 요약 층	전체를 훑어야 함
지난 분기 매출 합계는?	데이터베이스 질의(SQL)	세고 더하는 일이기 때문
이 보고서를 처음부터 끝까지 평가해 줘	긴 문맥	한 문서 전체를 읽는 일이기 때문
A사와 B사의 관계와 관련 쟁점은?	그래프 탐색 + 원문 검색	관계로 찾고 원문으로 확인
관련 정보를 조사해 종합 보고서로 써 줘	에이전트형 탐색	여러 단계를 거쳐야 함

하나의 질문에 여러 길을 동시에 쓰는 경우도 많습니다. 그래프로 관계를 찾고, 그 관계에 연결된 계약 원문을 하이브리드 검색으로 확인한 뒤, 금액은 데이터베이스 질의로 검증하는 식입니다.

전체 흐름을 한 문장으로 정리하면 이렇습니다. **문서형 자료는 미리 정리해 두고, 데이터베이스와 업무 앱은 원본을 그때그때 조회하며, 에이전트가 질문 종류에 맞는 도구를 골라 근거를 모은 뒤 출처와 함께 답합니다.**

10.3 기술 구성 — 작게 시작해 필요할 때 키우기

처음부터 대규모 시스템을 만들 필요는 없습니다.

기능	소규모 시작안	확장안
원문 저장	파일 저장 공간(Object Storage)	판본 관리가 되는 저장 공간
꼬리표·관계 정보	포스트그레스큐엘(PostgreSQL) 데이터베이스	PostgreSQL + 사건 기록(Event Log)
의미 검색	피지벡터(pgvector — PostgreSQL에 벡터 검색을 더한 확장)	전용 벡터 데이터베이스 또는 분산 검색

기능	소규모 시작안	확장안
키워드 검색	PostgreSQL 전체 글 검색(FTS)	엘라스틱서치(Elasticsearch) / 오픈서치(OpenSearch)
그래프	관계 표(테이블)	네오포제이(Neo4j) 같은 그래프 데이터베이스
위키	마크다운 + 데이터베이스	판본·승인 기능이 있는 위키
문서 해석	파이썬 기반 해석 도구	문서 종류별 전용 흐름
작업 처리	작업 대기열(큐)	카프카(Kafka) 같은 사건 기반 처리
평가·관측	평가 집합과 기록	자동 회귀 평가와 추적(Tracing)

참고할 점 두 가지. PostgreSQL의 전체 글 검색은 가벼운 시작용으로 쓸 수 있고 pgvector와 결합할 수 있지만, 그 순위 방식 자체가 BM25 알고리즘인 것은 아닙니다. 그리고 그래프 저장소도 처음부터 Neo4j가 필요한 것은 아닙니다. 관계의 양과 탐색 깊이가 작으면 PostgreSQL의 관계 표로 시작하고, 관계 질의가 복잡해질 때 전문 그래프 데이터베이스로 넓히면 됩니다. 한국어는 엘라스틱서치의 노리(Nori) 분석기처럼 한국어 형태소 분석과 사용자 정의 사전을 지원하는 도구가 유리합니다.

---

11. 방법론 비교와 출발점 고르기

11.1 한눈에 보는 비교

방법	가장 잘 맞는 질문	적정 규모	자료 변경	구축 난이도	질문당 비용·속도	대표 약점
긴 문맥 + 프롬프트 캐싱(CAG 포함)	모든 종류	수십만 토큰 이하	드물게	매우 쉬움	토큰은 많지만 캐시로 완화	길수록 정확도 저하, 규모 상한
하이브리드 검색 + 문맥 기반 검색 + 재정렬	한 곳에서 찾는 사실	제한 없음	잦아도 됨(덧붙여 갱신)	중간	낮음, 빠름	전역·다중 홉 질문
그래프 계열(변종별 차이 큼)	전역 요약, 다중 홉, 관계 추론	중간~대규모	드물게(라이트RAG는 덧붙이기 가능)	높음	변종마다 큰 차이	단순 사실 질문, 대상 품질 의존
엘엘엠 위키(LLM Wiki)	종합·탐구형, 쌓이는 지식	단일 주제 수백	꾸준한 추가	쉬움	낮음~중간	규모·권한·오류 누적

방법	가장 잘 맞는 질문	적정 규모	자료 변경	구축 난이도	질문당 비용·속도	대표 약점
		문서 이하				
에이전트형 검색 + MCP	원본 시스템의 최신 자료, 여러 단계 작업	제한 없음	실시간	중간 (도구 설계)	느리고 토큰 많음	지연, 매번 반복되는 탐색
텍스트-투-SQL(Text-to-SQL)	합계, 숫자, 기간 비교	제한 없음	실시간	중간	빠름	구조 이해 오류, 잘못된 질의
페이지 그림 검색 (ColPali 계열)	발표 자료, 스캔, 도표 많은 문서	중간	보통	중간	중간, 저장 공간 큼	저장 비용, 글자 정밀 검색 약함
미세 조정(Fine-tuning)	형식·말투·판단 습관	해당 없음	재훈련 필요	학습 비용 발생	빠름	자주 바뀌는 사실, 출처 제시 불가

11.2 규모별 권장 출발점

개인·소규모 — 문서 수백 건 이하, 토큰 20만 내외 긴 문맥 또는 엘엘엠 위키부터 시작합니다. 벡터 저장소부터 세우는 것은 대개 순서가 틀렸습니다. 마크다운 환경(예: 옵시디언, Obsidian)에 에이전트를 붙여 위키를 유지하게 하면, 검색 설비 없이도 출처를 댈 수 있는 답변까지 얻을 수 있습니다. 규모가 이를 넘기 시작하면 고도화된 검색 방식을 얹습니다.

팀·제품 — 문서 수천~수만 건, 매일 질문이 있는 경우 고도화된 검색 방식이 기본값입니다. 구조를 살리는 나누기 → 문맥 기반 검색 → 하이브리드 검색 → 재정렬 → 출처 강제. 여기에 평가 집합과 사용자 반응 수집을 처음부터 붙입니다. 관계형 질문의 비중이 실제로 측정해 보니 높을 때만 그래프 층을 추가합니다.

대규모·규제 분야 — 법률, 의료, 금융, 정책 고도화된 검색 + 구조화 자료 도구 + 권한 거르기 + 감사 기록이 필수 조합입니다. 에이전트 길 나누기는 어려운 질문 경로에만 제한적으로 적용하고, 모든 답변에 출처 검증을 겁니다.

11.3 다섯 가지 질문으로 정하는 순서

1. 전체 자료가 문맥에 들어가고 잘 안 바뀌는가? → 그렇다면 긴 문맥부터.
2. 질문이 대부분 단일 문서 안의 사실 찾기인가? → 그렇다면 고도화된 검색 방식.
3. 관계·다중 홉·전역 요약 질문이 핵심 가치인가? → 그렇다면 그래프 또는 위키의 연결 구조를 검토.
4. 지식이 개인의 쌓이는 노트인가, 조직의 공식 문서인가? → 전자면 위키, 후자면 검색 + 관리 체계.

5. 문제는 지식이 아니라 AI의 행동인가? → 그렇다면 미세 조정이나 지시 설계를 먼저.

참고 연구도 같은 방향을 가리킵니다. 2025년 국제 학회 ICML의 라라(LaRA) 연구는, 검색 방식과 긴 문맥 중 어느 쪽이 유리한지는 모델, 문서 길이, 검색 특성, 질문 종류에 따라 달라진다고 결론 내렸습니다. 어느 한쪽이 항상 이기는 것이 아닙니다.

---

12. 평가 — 측정할 수 없으면 개선할 수 없습니다

평가는 두 층으로 나눕니다. 섞어서 재면 어디가 고장인지 알 수 없습니다. **찾는 데 성공했는데 답이 틀린 것인지, 찾는 것 자체가 실패한 것인지**를 구분하는 것이 출발점입니다.

12.1 무엇을 잴 것인가

평가 층	잴 것	쉬운 뜻
수집·해석	문서 누락, 표·숫자 추출 정확성	재료가 제대로 들어왔는가
검색	재현율@K(Recall@K), 정밀도, 평균 상호 순위(MRR), 순위의 질(nDCG)	상위 K개 안에 정답 근거가 들어왔는가, 얼마나 위에 있는가
문맥	필요한 증거의 충분함, 중복	건넨 자료가 답하기에 충분하고 군더더기가 없는가
생성	충실도(Faithfulness), 답변 적합성	답의 주장이 근거로 뒷받침되는가, 질문에 답하고 있는가
출처	출처가 실제 주장을 뒷받침하는가	출처 표시가 겉모양뿐이 아닌가
전역 추론	여러 문서의 주제·관계 포괄성	전체를 아우르는 질문에 제대로 답하는가
보안	권한 없는 자료 노출 여부	보지 말아야 할 문서가 새지 않았는가
운영	95번째 백분위 지연(P95 — 100번 중 95번째로 느린 응답 시간), 질문당 비용, 최신성	느린 경우에도 쓸 만한가, 비용이 감당되는가

오픈소스 평가 도구 레이가스(RAGAS)는 이 중 충실도(Faithfulness), 답변 관련성(Answer Relevancy), 문맥 정밀도(Context Precision), 문맥 재현율(Context Recall) 네 지표를 표준적으로 제공합니다. 에레스(ARES) 연구도 평가를 문맥 관련성, 답변 충실도, 답변 관련성으로 나눠 다룹니다.

진단을 읽는 법은 이렇습니다. 충실도만 낮으면 답 쓰기 문제, 재현율이 낮으면 찾기 문제, 정밀도가 낮으면 재정렬·거르기 문제입니다. 채점은 AI를 판정자로 쓰되, 일부 표본은 사람이 직접 확인해 판정자 자체가 믿을 만한지

점검하는 것이 좋습니다.

12.2 여섯 가지 방식을 같은 시험으로 견주기

같은 자료와 같은 질문으로 아래 여섯 방식을 비교하면, 설계 선택의 증거가 만들어집니다. 처음부터 전체 자료에 적용할 필요는 없습니다. 대표 질문 100~200개면 충분합니다.

시험	구성
A	긴 문맥(Long Context)
B	의미 검색만(Vector RAG)
C	하이브리드 검색(Hybrid Search)
D	문맥 기반 검색 + 재정렬(Contextual Retrieval + Reranking)
E	하이브리드 + 부모-자식(Hybrid + Parent-Child)
F	그래프·위키 확장형(Graph / Wiki)

질문 종류에는 단일 사실, 정확한 식별자(제품 번호 등), 여러 문서의 관계, 전체 경향, 수치 합계, 서로 어긋나는 문서, 답이 없는 질문을 반드시 포함합니다.

복잡한 구조를 도입하기 전에 확인할 것도 정해져 있습니다. 단순 하이브리드 방식보다 얼마나 정확해졌는가, 특히 어떤 종류의 질문이 좋아졌는가, 추가로 드는 저장·갱신 비용은 얼마인가, 질문당 시간과 비용이 얼마나 늘었는가, 유지의 복잡함을 감당할 가치가 있는가. **가장 좋은 구조는 가장 복잡한 구조가 아니라, 실제 질문에 대한 정확함과 운영 비용의 균형이 가장 좋은 구조입니다.**

12.3 평가를 운영하는 원칙

- 실제 사용자 질문에서 뽑은 평가 집합 50~100개 이상을 운영 전에 만듭니다.
- 나누기 방식, 임베딩 모델, 가져올 개수, 지시문 중 무엇 하나라도 바꾸면 평가 집합을 다시 돌립니다. 평가 없는 변경은 개선이 아니라 주사위 던지기입니다.
- 운영 중에는 실패한 질문을 모으는 통로를 둡니다. '답을 찾지 못함' 기록과 사용자 반응이 다음 평가 집합의 재료이자, 다음에 고칠 곳의 우선순위 목록입니다.
- 공개된 시험 수치를 내 시스템의 수치로 착각하지 않습니다. 공개 수치는 방향(어떤 층이 어떤 실패를 줄이는가)을 알려 줄 뿐, 크기는 내 자료로 재야 합니다.

---

13. 운영과 보안 — 시범용과 실제 서비스의 차이

- **최신 상태 유지** — 문서가 바뀌면 해당 조각만 골라 다시 저장합니다(증분 색인, Incremental Indexing). 문맥 기반 검색을 쓰면 문맥 설명도 함께 다시 만듭니다.

- **지운 것은 모든 곳에서 지우기** — 삭제 전파(Deletion Propagation)라고 합니다. 원본에서 지운 문서가 검색 저장소에 남아 있으면 폐기된 규정으로 답합니다. 벡터, 그래프, 위키, 캐시 모두에서 지워졌는지 확인해야 합니다.
- **판본과 유효 기간** — 최신 문서와 과거 문서를 구분하는 정보(판본, 유효 시작·종료 시점)를 관리합니다. 그래프의 관계도 마찬가지입니다.
- **권한은 찾기 단계에서** — 사용자 권한을 검색 조건으로 적용합니다. 답을 만든 뒤에 검열하는 방식은 이미 늦습니다.
- **출처 추적(Provenance)** — 지식·관계·요약이 어느 원문에서 왔는지 추적할 수 있게 합니다.
- **비용 구조** — 비용은 저장할 때(임베딩 + 문맥 생성, 한 번)와 질문할 때(검색 + 재정렬 + 답 생성 토큰, 반복)로 나뉩니다. 프롬프트 캐싱이 양쪽 모두에서 가장 큰 절감 수단입니다. 질문 비용이 문제면 가져오는 후보 수부터 줄여 평가로 확인합니다.
- **속도** — 재정렬 후보 수와 에이전트 반복 횟수가 속도를 좌우합니다. 단계별 시간 한도를 정하고, 넘으면 대체 경로로 돌아가게 합니다.
- **기록 남기기(Observability)** — 질문, 다시 쓴 질문, 찾은 조각 번호, 재정렬 점수, 최종 출처를 기록합니다. 이 기록이 없으면 '왜 그렇게 답했나'에 영원히 답할 수 없습니다.
- **사람 검토** — 법률·의료·재무처럼 중요한 결과는 적절한 사람 검토를 거치게 합니다.

13.1 문서 속에 숨은 지시문 — 프롬프트 주입(Prompt Injection)

검색된 문서 안에 '이전 지시를 무시하라' 같은 문장이 들어 있을 수 있습니다. 이것을 AI가 명령으로 착각해 따르면, 악의적인 문서가 시스템 전체를 조종할 수 있습니다. 국제 보안 단체 OWASP의 검색 증강 생성 보안 안내서는, 원문 수집부터 저장, 검색, 답변, 외부 도구 실행까지 새로운 공격 지점이 생긴다고 설명합니다.

방어 원칙은 명확합니다. **찾아온 문서는 언제나 '자료'로만 다루고, '명령'으로 실행하지 않습니다.** 외부 웹이나 이메일을 수집하는 시스템이라면 특히 중요합니다. 권한 없는 자료가 다른 사용자에게 노출되는 문제도 같은 수준으로 다뤄야 합니다.

13.2 한국어 자료의 운영 점검

조사·어미 때문에 키워드 검색 정확도가 흔들리므로 형태소 분석 기반 키워드 검색 설정을 점검하고, 임베딩 모델은 한국어 평가 집합으로 직접 비교합니다. 한·영 혼합 문서에서는 두 언어 질문을 모두 시험합니다. 회사 줄임말과 사람 이름, 전문 용어, 기술 코드·제품 번호·계약 번호의 정확한 찾기도 별도로 확인합니다.

---

14. 단계별 구축 로드맵 — 가장 단순한 기준선부터

처음부터 모든 기술을 쌓지 마세요. 가장 단순한 기준선을 평가 집합으로 재고, 실패 종류에 맞는 기술만 하나씩 더해 가는 것이 비용 대비 가장 안전합니다.

0단계 — 진단과 평가 집합

자료 분량을 토큰으로 환산하고, 실제 질문 50~100개와 정답 위치를 모읍니다. 질문 종류(한 곳 찾기·전역·합계·실시간)의 비율을 확인합니다. 다음 단계로 넘어가는 조건은 평가 집합 확보입니다.

1단계 — 증거 기반 다지기

문서 해석, 원본 보존, 꼬리표 붙이기, 하이브리드 검색, 기본 출처 기능을 만듭니다. 자료가 수십만 토큰 이하이면 통째로 넣고 프롬프트 캐싱을 쓰는 길도 함께 시험합니다.

완료 기준: 주요 질문에 대해 적절한 원문 위치를 찾고 출처를 댈 수 있음. 평가 집합으로 실패율을 재고, 실패를 찾기 실패와 답 쓰기 실패로 나눕니다.

2단계 — 문맥을 살리는 검색

구조를 따르는 나누기, 부모-자식, 문맥 기반 검색, 재정렬을 차례로 시험합니다. 매번 다시 재는 것이 중요합니다.

완료 기준: 1단계보다 찾기 실패와 답변 오류가 측정 가능하게 감소함. 찾기 실패가 많으면 문맥 기반 검색과 재정렬을, 답 쓰기 실패가 많으면 출처 강제와 근거-답변 일치 검증을 더합니다.

실패 원인이 문맥 손실이면 문맥 기반 검색을, 경계 손실이면 부모-자식 구조를, 질문 쪽 문제이면 질문 다시 쓰기·늘리기·가상 답안을 먼저 붙입니다. 마지막 방법은 저장소를 건드리지 않으므로 가장 쌉니다.

3단계 — 질문 종류별 길 나누기

합계 질문은 데이터베이스 질의로, 전역·연결 질문은 그래프나 위키 요약 층으로, 실시간 자료는 MCP 도구로 보냅니다. 자주 등장하는 대상과 관계부터 뽑고, 주제별 위키를 만듭니다. 전면적인 그래프 생성보다 가치가 높은 지식 영역부터 시작하세요.

완료 기준: 여러 문서 관계 질문과 전체 경향 질문에서 성능 향상이 확인됨.

4단계 — 똑똑한 조율과 에이전트화

질문 길잡이(Query Router), 에이전트형 검색, 데이터베이스·업무 도구, 자동 검증, 증분 지식 갱신을 더합니다. 검색을 에이전트의 도구로 만들어 '찾고, 읽고, 부족하면 다시 찾는' 반복 탐색을 허용하고, 복잡한 조사는 보조 에이전트(서브에이전트, Sub-agent)에게 나눠 맡겨 요약만 돌려받게 합니다.

완료 기준: 다양한 질문 종류를 비용과 정확도에 맞는 길로 처리함. 재시도 횟수 상한, 시간 한도, 근거가 부족할 때 멈추는 규칙을 명시해야 합니다.

5단계 — 운영 순환

기록에서 실패 사례를 뽑아 평가 집합에 계속 추가하고, 증분 저장·삭제 전파·정기 점검을 일정에 넣습니다.

개인·소규모 팀을 위한 최소 구성

개발 설비 없이 시작하려면 **에이전트 도구 + 마크다운 폴더(엘엘엠 위키 패턴)**가 가장 빠른 출발점입니다. 원본은 `raw/`에 넣고, 규칙 파일에 페이지 양식과 '원본 수정 금지, 출처 연결 필수' 규칙을 적은 뒤 수집·질의·점검

을 반복합니다. 위키가 수백 문서를 넘어 목차만으로 찾기 어려워지면 qmd 같은 지역 하이브리드 검색을 붙이고, 그보다 커지거나 여러 사람이 다른 권한으로 써야 하면 2단계 검색 구성으로 옮기면 됩니다.

---

15. 설계 체크리스트

설계 검토 자리에서 그대로 쓸 수 있는 점검표입니다.

자료와 나누기

- [ ] 해석 도구가 표·제목·목록 구조를 보존하는가
- [ ] 조각만 읽고 뜻을 알 수 있는가(한 문장 검사)
- [ ] 자료 종류별로 나누기 규칙이 다른가
- [ ] 모든 조각에 출처·날짜·권한 정보가 있는가
- [ ] 옛 판본·중복 문서 처리 규칙이 있는가
- [ ] 원본, 뽑아낸 글, 표 구조, 원본 위치가 연결돼 있는가

저장과 찾기

- [ ] 의미 검색용과 키워드 검색용을 모두 만들었는가
- [ ] 조각에 문맥 설명을 붙이는 방안을 검토했는가(문맥 기반 검색)
- [ ] 두 검색을 합치는 방식이 순위 기반(RRF)인가
- [ ] 재정렬 단계와 최종으로 건넬 개수가 정해져 있는가
- [ ] 권한 거르기가 찾기 단계에서 적용되는가
- [ ] 질문 다시 쓰기·늘리기 중 최소 하나가 있는가
- [ ] 한국어 형태소 분석과 한·영 혼합 질문을 시험했는가

답 쓰기와 신뢰

- [ ] 출처가 강제되고, 출처-내용 일치 검증이 있는가
- [ ] 근거가 없을 때 '모른다'고 답하는 길이 있는가
- [ ] 가장 관련 높은 문맥의 배치와 중복 제거 규칙이 있는가
- [ ] 사실과 추론이 답 안에서 구분되는가

평가와 운영

- [ ] 종류를 골고루 담은 평가 집합 50~100개가 있는가
- [ ] 찾기 지표와 답 쓰기 지표를 나눠 재는가
- [ ] 여섯 가지 방식(A~F)을 같은 시험으로 견줘 봤는가
- [ ] 증분 갱신과 삭제 전파 절차가 있는가
- [ ] 질문-검색-출처 기록이 남는가
- [ ] 실패 질문을 다음 개선으로 잇는 순환이 있는가

- [ ] 문서 속 지시문(프롬프트 주입)을 자료로만 다루는가

---

16. 흔한 안티패턴 — 하지 말아야 할 것

안티패턴(Anti-pattern)은 그럴듯해 보이지만 반복해서 실패를 만드는 나쁜 습관입니다.

1. **평가 집합 없이 조각 크기부터 조정하기.** 측정 없는 최적화는 미신입니다.
2. **의미 검색 하나로 모든 질문 받기.** 코드·이름·날짜 질문에서 반드시 뚫립니다. 하이브리드가 기본입니다.
3. **미세 조정으로 사실 가르치기.** 낡고, 출처를 댈 수 없고, 권한을 걸 수 없습니다.
4. **긴 문맥을 대규모 지식 자료의 대체재로 쓰기.** 비용은 자료 크기에 비례해 늘고, 정확도는 문맥 부패와 중간 소실로 무너집니다.
5. **그래프 방식을 유행으로 도입하기.** 관계 질문이 핵심이 아닐 때는 3~5배 비용을 내고 단순 조회만 느려집니다. 단순 사실 질문에서는 기본 방식보다 못하다는 연구 결과도 있습니다.
6. **검색 성공을 답변 성공으로 착각하기.** 상위 목록에 들어간 것과 AI가 그것을 근거로 답한 것은 별개로 검증합니다.
7. **권한을 답을 만든 뒤에 막기.** 찾기 단계의 거르기가 유일하게 안전한 위치입니다.
8. **실패를 모델 탓으로만 돌리기.** 대부분의 실패는 해석·나누기·저장 단계에서 이미 정해져 있습니다.
9. **AI가 만든 위키와 요약을 원본처럼 믿기.** 파생 지식은 원본의 권위를 물려받지 않습니다. 출처로 되짚을 수 있어야 합니다.
10. **생성된 문맥 설명을 원문과 한 덩어리로 보관하기.** 따로 보관하고 출처는 원문으로 연결해야, 설명이 틀렸을 때 바로잡을 수 있습니다.

맺음말 — 좋은 시스템은 '똑똑한 모델'이 아니라 '좋은 증거 공급망'입니다

처음 제기된 문제의식 — 검색 증강 생성(RAG)은 생각만큼 쉽지 않고, 나누는 과정에서 문맥이 사라지며, 대안이 여럿 존재한다 — 은 정확합니다. 지난 2년의 발전은 그 문제의식을 하나씩 제도로 바꾼 과정이었습니다.

- 문맥 기반 검색(Contextual Retrieval)은 조각에 문맥을 되돌려 주는 것으로 답했습니다.
- 그래프 방식(GraphRAG)은 관계 그 자체를 저장 대상으로 삼는 것으로 답했습니다.
- 엘엘엠 위키(LLM Wiki)는 아예 검색 대신 미리 정리(컴파일)를 택하는 것으로 답했습니다.
- 에이전트형 검색(Agentic Retrieval)은 찾기를 한 번의 동작이 아니라 판단의 반복으로 바꾸는 것으로 답했습니다.
- 캐시 증강 생성(CAG)과 긴 문맥은, 작을 때는 아예 찾지 않는 것으로 답했습니다.

설계자의 일은 이 이름들을 외우는 것이 아닙니다. 내 질문이 어디서 실패하는지를 측정하고, 그 실패에 맞는 층을 하나씩 쌓는 것입니다. 측정 → 제대로 된 기본형 → 문맥 되살리기 → 질문 종류별 확장 → 선별적인 에이전트화. 이 순서를 지키면 방법론의 유행이 바뀌어도 시스템은 무너지지 않습니다. 바뀌는 것은 층이지, 원칙이 아니기 때문입니다.

장기적으로 가장 큰 가치는 새로운 검색 알고리즘 자체보다, **자료를 검증 가능한 지식으로 바꾸고 지속적으로 관리하는 능력**에 있습니다. 이것은 기술 선택을 넘어, 자료 다루기·지식 정리·검색·추론·운영 관리가 결합된 문제입니다.

참고 자료

수치는 각 자료가 공개한 시험 기준이며, 방향을 판단하는 근거로 제시한 것입니다. 실제 도입 효과는 자체 평가 집합으로 다시 재야 합니다.

핵심 방법론

- 앤트로픽(Anthropic), 문맥 기반 검색 소개 — Introducing Contextual Retrieval (2024.9.19). 실패율 수치(5.7% → 2.9% → 1.9%), 비용 추산, 20만 토큰 기준: https://www.anthropic.com/research/contextual-retrieval
- 앤트로픽(Anthropic), 문맥 임베딩 안내서 — Contextual Embeddings Cookbook. 구현 예제: https://platform.claude.com/cookbook/capabilities-contextual-embeddings-guide
- 마이크로소프트 리서치(Microsoft Research), GraphRAG 원 논문 — From Local to Global: A Graph RAG Approach to Query-Focused Summarization (2024.4): https://arxiv.org/html/2404.16130v2
- 마이크로소프트 리서치(Microsoft Research), LazyGraphRAG (2024.11). 저장 비용 0.1%, 전역 질문 비용 700배 이상 절감: https://www.microsoft.com/en-us/research/blog/lazygraphrag/
- 궈 외(Guo et al.), LightRAG (2024.10). 이중 수준 검색과 증분 갱신: https://arxiv.org/abs/2410.05779v3
- 사르티 외(Sarthi et al.), RAPTOR (ICLR 2024). 계층 요약 나무 검색: https://www.arxiv.org/abs/2401.18059
- 지나 AI(Jina AI), Late Chunking (2024.8). 임베딩 후 나누기 방식: https://jina.ai/news/late-chunking-in-long-context-embedding-models/
- 페이스 외(Faysse et al.), ColPali (ICLR 2025). 페이지 그림 기반 문서 검색: https://arxiv.org/abs/2407.01449

비교와 평가 연구

- 샹 외(Xiang et al.), GraphRAG-Bench: When to Use Graphs in RAG (ICLR 2026, 2026.2 개정). 과제별 비교, 프롬프트 팽창, 자료 규모별 안정성: https://arxiv.org/pdf/2506.05690
- LaRA (ICML 2025). 검색 방식과 긴 문맥의 비교 — 어느 쪽이 유리한지는 조건에 따라 다름
- 찬 외(Chan et al.), Don't Do RAG: Cache-Augmented Generation (WWW 2025). 캐시 증강 생성의 개념과 적용 조건: https://arxiv.org/abs/2412.15605v1
- 크로마(Chroma), Context Rot (2025.7.14). 입력 길이에 따른 모델 성능 저하: https://trychroma.com/research/context-rot

- 앤트로픽(Anthropic), Effective Context Engineering for AI Agents (2025.9.29). 필요할 때 가져오기, 혼합 전략, 메모리와 보조 에이전트: https://www.anthropic.com/research/effective-context-engineering-for-ai-agents
- ARES (NAACL 2024). 검색 증강 생성 품질 평가 방법 — 문맥 관련성, 답변 충실도, 답변 관련성
- 레이가스(RAGAS) 평가 프레임워크. 충실도, 답변 관련성, 문맥 정밀도, 문맥 재현율

위키와 에이전트형 검색

- 안드레이 카르파티(Andrej Karpathy), LLM Wiki 아이디어 파일 (2026.4). 원 설계 문서: https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f
- 임플리케이터(Implicator), Karpathy LLM Wiki 분석 (2026.4.14). 구조, 규모 한계, 오류 누적과 분리 권고: https://www.implicator.ai/karpathys-llm-wiki-got-16-million-views-the-code-isnt-the-point-2/
- 스타모프(Starmorph), How to Build Karpathy's LLM Wiki (2026.4.9). 세 가지 작업, 검색 방식과의 사용 구분: https://blog.starmorph.com/blog/karpathy-llm-wiki-knowledge-base-guide
- 쿠바 로구트(Kuba Rogut), RAG is dead, right? (AI Engineer Europe, 2026.4.9). 클로드 코드의 벡터 저장소 제거 배경: https://ai.engineer/talks/UM6sFg_jdlE-rag-is-dead-right-kuba-rogut
- 토비 뤼트케(Tobi Lütke), qmd. 마크다운용 지역 하이브리드 검색: https://github.com/tobi/qmd

보안

- OWASP, RAG Security Cheat Sheet. 실제 운영 보안 요구 사항 — 프롬프트 주입, 권한 없는 자료 노출 등

> 이 보고서는 연구 결과와 기술 문서에 근거한 설계 제안입니다. 특정 조직의 실제 자료에서 성능을 검증한 결과는 아니므로, 구현에 앞서 작은 자료 집합을 이용한 비교 실험이 필요합니다.