현업에서 사용하는 RAG

RAG Survey 논문 속 Retrieve-Read 구조와 실제 서비스 아키텍처, 평가 루프, 운영 시스템

최지안

RAG Survey를 출발점으로 삼되, 발표의 중심은 프로덕션 RAG의 설계 판단과 실패 대응

20240600

TABLE OF CONTENTS

발표 목차

1부
왜 RAG인가

2부
RAG 이론과 패러다임 진화

3부
파이프라인 전체 설계

4부
색인과 검색

5부
고급 아키텍처

6부
생성과 평가

7부
운영과 배포

8부
마무리

1부. 왜 RAG인가

ARGUMENT

현업 RAG의 등장

좋은 답변은
커다란 DB만으로 나오지 않는다.

01 · RETRIEVAL

찾아야 할 문맥을 잘 찾아야 한다

질문 재작성, 소스 라우팅, 하이브리드 검색, 재순위화 등

02 · GROUNDING

답변이 근거에 묶여야 한다

문맥 압축, 인용, 근거 없는 답변 거절, 구조화 출력 등

03 · OPERATION

깨졌을 때 알 수 있어야 한다

평가셋, 로그, 드리프트, 권한, 최신성, 비용과 지연 시간.

이론 배경 · RAG의 원점

LLM에는 두 가지 구조적 한계가 있다

한계 1 · 지식 경계

LLM은 학습 시점까지의 정보만 파라미터에 저장합니다. 학습 이후 갱신된 정보나 기업 내부 문서는 모릅니다.

한계 2 · 환각 (Hallucination)

LLM은 질문이 지식 경계 바깥에 있을 때 존재하지 않는 내용을 그럴듯하게 만들어냅니다. 자연스럽기 때문에 사용자가 알아채기 어렵습니다.

LEWIS ET AL. (2020) — RAG의 해결책

필요한 순간에 외부 문서를 검색해서 프롬프트에 붙인다.

파라미터를 건드리지 않고도 최신 지식을 모델에 주입할 수 있다는 점이 핵심 장점입니다.

2부. RAG 이론과 패러다임 진화

이론 배경 · RAG 패러다임

RAG의 세 가지 발전 단계

NAIVE RAG

색인 → 검색 → 생성

가장 단순한 3단계 파이프라인. 빠르게 구축할 수 있지만 파싱 오류, 표현 불일치, 충실성 문제에 취약합니다.

ADVANCED RAG

Pre + Core + Post

Pre-retrieval(쿼리 최적화)과 Post-retrieval(재순위화·압축)을 추가해 검색 품질을 높인 구조입니다.

MODULAR RAG

독립 모듈의 재조합

검색, 메모리, 라우팅, 융합 등 기능별 모듈을 독립적으로 구성하고 재조합할 수 있습니다. 현업 시스템과 가장 가깝습니다.

이 세 패러다임이 오늘 발표의 뼈대입니다. Naive → Advanced → Modular 순서로 개선됩니다.

PRODUCTION REQUIREMENTS

현업 RAG가 만족해야 하는 조건

1
정확성

관련 문서를 찾는 것과 답변이 그 문서에 충실한 것은 별개의 문제입니다.

2
최신성

문서 변경, 삭제, 버전 갱신이 색인에 반영되어야 합니다.

3
권한

검색 결과 자체가 사용자 권한과 tenant 경계를 넘어가면 안 됩니다.

4
지연 시간

검색, rerank, LLM 호출이 직렬로 붙기 때문에 p95 latency가 쉽게 커집니다.

5
비용

임베딩, reranker, 긴 프롬프트, 재시도는 모두 누적 비용이 됩니다.

6
관측 가능성

나쁜 답변이 나왔을 때 검색 실패인지, 생성 실패인지 분해할 수 있어야 합니다.

BASELINE

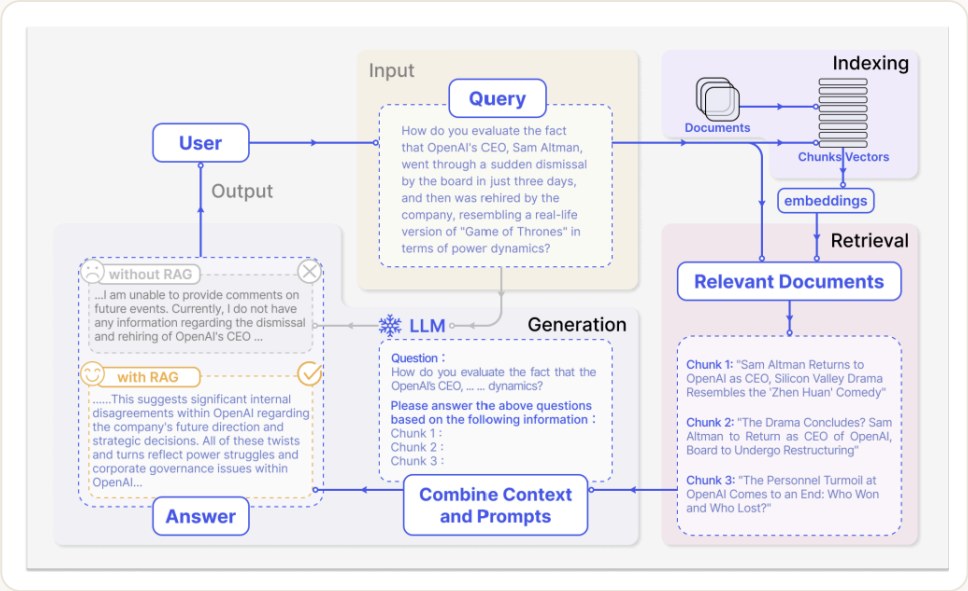
Naive RAG는 좋은 아이디어지만, 운영 요구사항을 완전히 만족하지 못한다

기본 구조

문서를 청크로 나누고 임베딩한 뒤, 질문과 가까운 청크를 가져와 프롬프트에 붙입니다.

현업에서 Naive RAG가 문제 생기는 예시

- PDF 표·이미지·각주가 파싱 중 손상됨(색인 단계 문제)
- 질문 표현과 문서 표현이 어긋남(검색 단계 문제)
- Top-k에 노이즈가 섞이고 핵심 근거가 빠짐(검색 단계 문제)
- 답변은 자연스럽지만 근거와 맞지 않음(생성 단계 문제)



Naive RAG의 색인화 · 검색 · 생성 흐름

3부. 파이프라인 전체 설계

FROM NAIVE TO ADVANCED

현업 RAG의 첫 번째 개선은 검색 전·후를 두껍게 만드는 것

핵심 검색 품질은 검색기 하나가 아니라 전처리와 후처리의 합으로 결정됩니다.

이론 배경 · 질의 최적화

질문을 검색 공간으로 이동시키는 두 가지 방법

HYDE — GAO ET AL. (2022)

HyDE (Hypothetical Document Embeddings)

사용자의 질문을 그대로 임베딩하는 대신, LLM으로 "이 질문에 대한 **가상의 답변 문서**"를 먼저 생성하고, 그 가상 문서를 임베딩해서 검색합니다.

원리: 실제 문서는 "답변 형태"로 쓰여 있고 사용자 질문은 "질문 형태"입니다. 두 형태는 임베딩 공간에서 거리가 멉니다. 가상 답변을 생성해 실제 문서와 같은 벡터 공간으로 이동시킵니다. 가상 답변의 사실 여부는 중요하지 않습니다.

QUERY REWRITING — MA ET AL. (2023)

Query Rewriting

소규모 LLM을 사용해 사용자 질문을 검색에 최적화된 형태로 재작성합니다.

Before: "그건 어떻게 돼?"

After: "A 제품의 반품 정책은 무엇인가?"

검색기가 대화 맥락을 이해하지 못해도 올바른 문서를 찾을 수 있게 됩니다.

4부. 색인과 검색

INGESTION

문서 수집 단계에서 이미 답변 품질이 결정된다

검색이 안 되는 문제는 주로 문서 파이프라인 문제

파싱

PDF, 표, 이미지 OCR, 머리말/꼬리말, 페이지 번호, 다단 문서, 코드 블록을 어떻게 보존할지 결정합니다.

정규화

중복 제거, 제목 계층 복원, 문서 단위 버전 관리, 언어·도메인별 토크나이징.

메타데이터

source, page, section, timestamp, author, permission, tenant, retention policy를 청크와 함께 저장합니다.

갱신

CHUNKING

청킹은 "몇 토큰으로 자를까"보다 더 큰 문제다

GRANULARITY

작으면 정밀하지만 맥락이 사라진다

문장 단위는 정확한 근거를 찾기 좋지만, 답변에 필요한 주변 설명이 부족할 수 있습니다.

CONTEXT

크면 맥락은 살지만 노이즈가 들어온다

큰 청크는 생성에는 유리하지만 검색 점수가 문서 전체 평균에 끌릴 수 있습니다.

STRUCTURE

문서 구조를 색인에 반영한다

parent-child retrieval, heading path, table summary, page span을 같이 저장합니다.

현업에서 자주 쓰는 타협

검색 단위	생성 단위	필터 단위
작은 청크 또는 문단 요약	부모 문단, 섹션, 앞뒤 문맥	문서 권한, 날짜, 제품군, 버전

이론 배경 · 청킹 입도

청킹 단위의 스펙트럼과 Parent-Child 구조

논문의 입도 스펙트럼 — 세밀 → 거친 순

Token → Phrase → Sentence → **Proposition** → Chunk → Document

Proposition: 문서에서 원자적 사실(atomic claim)을 추출합니다. 예: "파리는 프랑스의 수도이다." 가장 정밀하지만 전처리 비용이 높습니다.

Parent-Child Retrieval

검색 시에는 작은 "child" 청크(문장 2~3개)로 정밀하게 일치를 찾고, LLM에게는 그 청크가 속한 "parent" 문단이나 섹션 전체를 컨텍스트로 제공합니다.

→ 검색 정밀도와 생성 맥락 풍부함을 동시에 얻는 타협점

두 마리 토끼

검색 단위 (child): **작게** → 정밀한 매칭

생성 단위 (parent): **크게** → 풍부한 맥락

QUERY UNDERSTANDING

실제 사용자의 질문은 검색 질의가 아니다

그래서 현업 RAG는 검색 전에 질문을 다시 읽는다.

질의 최적화는 "더 똑똑한 프롬프트"라기보다, 어떤 검색기를 어떤 조건으로 호출할지 결정하는 라우팅 계층입니다.

Rewrite

대화 생략어, 지시어, 긴 질문을 독립 질의로 재작성합니다.

Route

문서 검색, SQL, 지식 그래프, 웹 검색, FAQ, 도구 호출 중 경로를 고릅니다.

Expand

multi-query, sub-query, HyDE로 표현 차이를 줄이고 회수율을 올립니다.

Constrain

제품 버전, 날짜, 사용자 권한, 문서 유형 같은 hard filter를 검색 전에 적용합니다.

이론 배경 · 질의 확장

Multi-Query와 Sub-Query: 표현 차이를 넘는 두 가지 방법

MULTI-QUERY EXPANSION

같은 질문, 다른 표현으로

하나의 질문에 대해 표현이 다른 여러 버전의 질의를 LLM으로 생성하고, 각각 검색한 뒤 결과를 합칩니다.

Q: "GPU 메모리 부족 오류 해결 방법"
① "CUDA out of memory error fix"
② "GPU OOM 에러 원인과 해결책"
③ "딥러닝 GPU 메모리 최적화 방법"

결과는 union → dedup → rerank 합니다.

SUB-QUERY DECOMPOSITION

복잡한 질문을 단순 질문들로

복잡한 질문을 여러 단순 질문으로 분해하고 각각 검색합니다.

Q: "A와 B의 성능 차이와 적합한 사용 사례를 설명해줘"
① A의 성능 특성은?
② B의 성능 특성은?
③ A의 사용 사례는?
④ B의 사용 사례는?

각각 검색한 뒤 통합해 종합 답변을 생성합니다.

RETRIEVAL STACK

검색은 보통 한 번의 벡터 검색으로 끝나지 않는다



왜 HYBRID가 자주 쓰이나

BM25는 고유명사·오류 코드·제품명처럼 정확한 문자열에 강하고, dense retrieval은 표현이 다른 의미적 유사성에 강합니다. 둘을 섞으면 long-tail 질의에서 안정성이 올라갑니다.

후보를 넓게, 문맥은 좁게

초기 검색은 recall을 우선하고, reranker와 dedup에서 precision을 회복합니다.

점수 결합은 정책이다

BM25 점수, vector score, freshness, permission, popularity, source reliability를 어떻게 섞을지 정해야 합니다.

이론 배경 · 검색 원리

BM25와 Dense Retrieval은 서로 다른 강점을 갖는다

DENSE RETRIEVAL — BI-ENCODER

의미적 유사성 검색

BERT 계열 모델로 질문과 문서를 각각 고차원 벡터로 인코딩합니다. 내적이나 코사인 유사도로 가장 가까운 문서를 찾습니다. 문서 벡터는 미리 계산해 저장합니다.

장점: "자동차를 고치다"와 "차량을 수리하다"를 같은 의미로 인식합니다.

ANN: HNSW, IVF 알고리즘으로 수백만 벡터에서 ms 단위 검색 (Faiss, Qdrant, Pinecone 등).

BM25 — SPARSE RETRIEVAL

정확한 문자열 매칭

TF-IDF를 개선한 알고리즘. 용어 빈도(TF)와 역문서 빈도(IDF)에 문서 길이 정규화를 추가합니다.

장점: "에러코드 E-0042"나 희귀 라이브러리 이름처럼 벡터 공간에서 표현되기 어려운 문자열을 정확히 찾습니다.

약점: 표현이 다르면 찾지 못합니다.

Hybrid = BM25 + Dense. 둘을 섞으면 long-tail 질의에서 안정성이 올라갑니다.

이론 배경 · 재순위화

Reranker(Cross-Encoder)와 RRF로 정밀도를 높인다

CROSS-ENCODER RERANKER

Two-Stage 검색 구조

Bi-encoder로 빠르게 수백 개의 후보를 뽑은 뒤, Cross-encoder(질문+문서를 함께 입력해 관련성 점수를 출력)로 정밀하게 순위를 매깁니다.

Bi-encoder
top 50~200
→
Cross-encoder
top 5~12

Cross-encoder는 더 정확하지만 후보 수에 비례해 비용이 늘어납니다. 후보를 미리 줄여야 합니다. (Cohere Rerank, BGE-Reranker 등)

RRF — RECIPROCAL RANK FUSION

점수 스케일이 달라도 합산

BM25 점수와 Vector 점수는 스케일이 달라 단순 합산이 어렵습니다. RRF는 각 시스템에서의 **순위**를 기반으로 결합합니다.

score = Σ 1/(k + rank_i)
(보통 k = 60)

점수 정규화 없이도 두 검색기의 결과를 효과적으로 합칩니다.

CONTEXT ASSEMBLY

검색된 문서를 그대로 넣으면 LLM이 더 잘 답하지 않는다

RERANK

중요한 근거를 앞뒤로 배치한다

긴 문맥에서 중간 정보가 약해지는 문제 때문에 배치 순서도 품질 변수입니다.

COMPRESS

문맥을 줄이되 근거는 남긴다

문장 선택, 요약, LLMLingua류 압축을 사용합니다.

DEDUPLICATE

같은 말을 여러 번 넣지 않는다

중복 문단이 많으면 모델이 반복하거나 중요도를 과대평가할 수 있습니다.



실무 팁: "가능한 한 많이 넣기"보다 "답변에 필요한 근거만 넣기"가 대체로 더 안정적입니다.

이론 배경 · 문맥 최적화

긴 문맥은 배치 순서와 압축으로 관리한다

LOST IN THE MIDDLE — LIU ET AL. (2023)

LLM은 긴 문맥의 중간을 무시한다

LLM은 긴 문맥에서 처음과 끝에 있는 정보는 잘 활용하지만 중간에 위치한 정보는 상대적으로 무시하는 경향이 있습니다. 문맥 길이가 길어질수록 심화됩니다.

✓ 가장 중요한 청크 → **프롬프트 맨 앞**
△ 보조 정보 → 중간
✓ 두 번째로 중요한 청크 → **프롬프트 맨 뒤**

LLMLINGUA — PAN ET AL. (2023)

토큰을 줄여도 LLM은 이해한다

작은 언어 모델을 사용해 문맥에서 정보량이 낮은 토큰을 제거합니다. 결과물은 사람이 읽기 어렵지만 LLM은 여전히 의미를 이해합니다.

효과: 프롬프트 토큰 수를 2~4배 줄이면서 답변 품질 유지. 비용과 지연 시간을 함께 줄입니다.

5부. 고급 아키텍처

MODULAR RAG

Modular RAG는 현업 시스템의 형태와 가장 가깝다

이론 배경 · MODULAR 패턴

Modular RAG: 고정된 Retrieve-Read 순서를 벗어나다

REWRITE-RETRIEVE-READ

검색 전에 쿼리 재작성 모듈을 삽입합니다. 현업에서 가장 먼저 추가하는 첫 번째 모듈입니다.

GENERATE-READ

외부 검색 대신 LLM이 자체 지식으로 가상 문맥을 생성한 뒤 답변합니다. HyDE와 연결됩니다.

RECITE-READ

LLM의 파라미터에서 관련 지식을 인출(recite)한 후 그것을 문맥으로 활용합니다.

DEMONSTRATE-SEARCH-PREDICT (DSP)

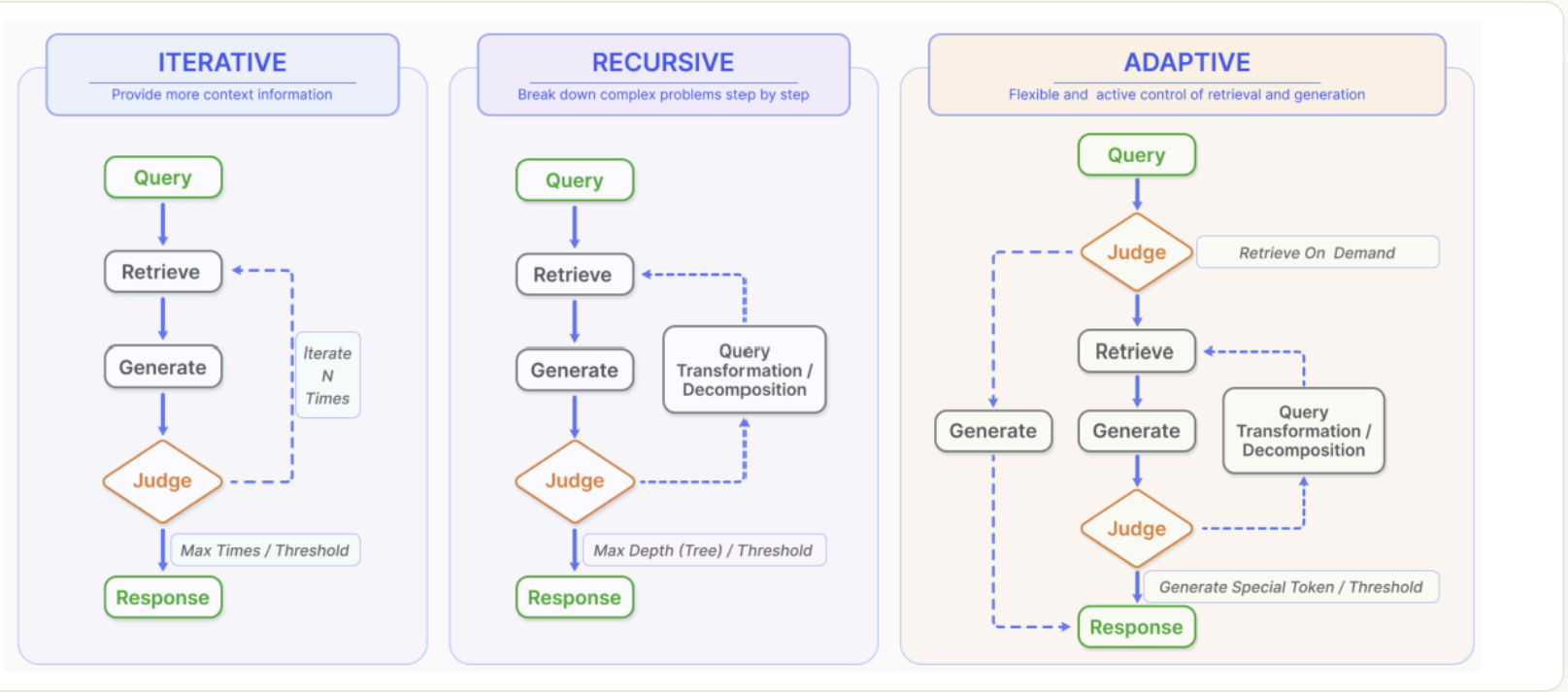
Few-shot 예시로 검색과 예측 단계를 반복적으로 조율합니다. 복잡한 추론 과제에 유효합니다.

핵심

모듈이 독립적이므로 특정 실패 지점만 교체하거나 개선할 수 있습니다. 파이프라인 순서도 필요에 따라 재조합할 수 있습니다.

RETRIEVAL ORCHESTRATION

복잡한 질문은 검색도 여러 번 해야 한다



Iterative

Recursive

Adaptive

6부. 생성과 평가

GENERATION

생성 단계의 목표는 "그럴듯함"이 아니라 "근거에 묶인 답변"이다

좋은 RAG 답변은 근거 밖으로 나가지 않는다.

검색 결과가 부족하면 거절하고, 근거가 충돌하면 충돌을 드러내며, 출처가 필요한 문장은 citation span에 연결해야 합니다.

Grounded prompt

"제공된 근거 안에서만 답하라"는 지시만으로는 부족하고, 인용·불충분 판단·출력 형식을 함께 정의합니다.

Abstention

검색 문맥에 답이 없으면 "모른다"라고 말하는 것이 품질입니다. 이를 평가셋에 포함해야 합니다.

Structured output

정책 답변, 보고서 초안, 고객지원 응답은 JSON schema나 템플릿으로 후처리 가능성을 높입니다.

EVALUATION

RAG 평가는 검색과 생성을 분리해서 봐야 한다

RETRIEVAL METRICS

올바른 근거가 후보에 들어왔는가

- Recall@k, Hit Rate, MRR, NDCG
- context precision / context recall
- source coverage, freshness, permission pass rate

GENERATION METRICS

근거를 바탕으로 답했는가

- faithfulness, answer relevance
- citation accuracy
- refusal quality, contradiction handling

실무 평가셋 구성

Golden QA

질문, 정답, 근거 문서 id/page/span을 함께 보관합니다.

Negative cases

답이 없는 질문, 권한이 없는 질문, 오래된 문서가 섞인 질문을 포함합니다.

Regression suite

청킹, 임베딩, reranker, 프롬프트를 바꿀 때마다 같은 질문으로 비교합니다.

Gao et al. (2024)는 문맥 관련성, 답변 충실성, 답변 관련성을 핵심 품질 점수로 정리합니다.

MONITORING

프로덕션에서는 "좋은 데모"보다 "나쁜 답변 추적"이 중요하다

TRACE

질의별 실행 경로

rewritten query, selected source, retrieved chunks, rerank score, prompt size, model output을 남깁니다.

FEEDBACK

사용자 신호

thumbs, citation click, copy, regenerate, escalation, manual correction을 평가셋 후보로 수집합니다.

ALERT

품질과 비용 알림

retrieval miss, empty context, hallucination judge, latency p95, token cost, index staleness를 모니터링합니다.

분석 단위

query intent → source → retrieval miss → generation failure → user impact

7부. 운영과 배포

SECURITY

사내 RAG는 검색 보안이 곧 모델 보안이다

모델이 모르는 정보도 검색이 가져오면 출력될 수 있다.

따라서 보안 경계는 생성 프롬프트가 아니라 검색 레이어와 인덱스 설계에서 먼저 잡아야 합니다.

ACL은 검색 전에 적용

검색 후에 숨기는 방식은 노출 위험과 랭킹 왜곡을 동시에 만듭니다.

프롬프트 인젝션은 문서에도 있다

문서 안의 "이전 지시를 무시하라" 같은 텍스트를 데이터로 취급하도록 경계를 분리합니다.

출처와 원문 접근을 분리

답변에 인용은 제공하되, 사용자가 원문을 볼 권한이 있는지 별도로 확인합니다.

감사 가능성

어떤 문서가 어떤 답변에 사용됐는지 추적할 수 있어야 합니다.

RAG VS FINE-TUNING

RAG와 Fine-tuning은 선택지가 아니라 역할이 다르다

RAG가 좋은 경우

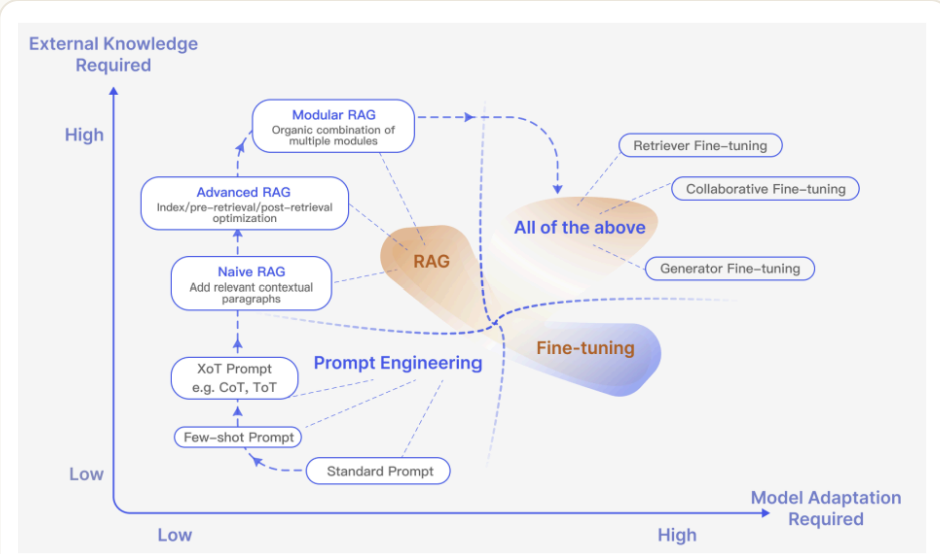
지식이 자주 바뀌고, 출처 검증이 필요하며, 사용자별 권한과 최신성이 중요할 때.

Fine-tuning이 좋은 경우

스타일, 형식, 도메인 작업 방식, 도구 사용 패턴을 모델 행동으로 내재화해야 할 때.

함께 쓰는 경우

검색기는 도메인 데이터로 맞추고, 생성기는 답변 형식·거절·인용 습관을 맞춥니다.



Prompt engineering, RAG, Fine-tuning의 위치 비교

이론 배경 · RAG VS FINE-TUNING

논문의 다섯 차원 비교

차원	RAG	Fine-tuning
지식 갱신	인덱스만 업데이트하면 됩니다	재학습이 필요합니다
해석 가능성	높음 — 근거 문서를 제시할 수 있습니다	낮음 — 지식이 가중치에 내재화됩니다
학습 비용	없음 (또는 소량)	높음
추론 지연	검색 지연이 추가됩니다	빠름
개인화	문서 수준 개인화 가능	행동 수준 개인화 가능

결론

동적 지식은 RAG · **고정된 행동 패턴**은 Fine-tuning · **둘 다 필요하면** 함께 씁니다.

LATENCY AND COST

RAG 품질을 올리는 모듈은 대부분 비용과 지연 시간을 늘린다

FAST PATH

간단한 질문은 짧은 경로

FAQ, 캐시, 작은 모델, 낮은 top-k로 응답하고 필요할 때만 비싼 경로로 보냅니다.

CANDIDATE BUDGET

Rerank 후보 수를 제한

초기 recall은 넓게 가져가되, cross-encoder/LLM rerank 입력 수는 제한합니다.

PROMPT BUDGET

문맥 토큰을 관리

중복 제거, 압축, 인용 span 선택으로 LLM 입력을 줄입니다.

운영 판단

모든 질문에 최고의 RAG 파이프라인을 태우는 것이 아니라,
질문 난이도와 위험도에 따라 검색 깊이를 조절하는 것이 실무적인 설계입니다.

PRODUCTION PLAYBOOK

현업 RAG 구축 순서

1
질문 수집

사용자가 실제로 물을 질문과 답이 없어야 하는 질문을 모읍니다.

2
근거 매핑

각 질문이 어떤 문서·페이지·문장으로 답해야 하는지 표시합니다.

3
색인 설계

청킹, 메타데이터, 권한, 갱신 정책을 먼저 정합니다.

4
검색 평가

LLM을 붙이기 전에 retrieval recall과 source coverage를 봅니다.

5
문맥 조립

rerank, dedup, compression, citation span을 튜닝합니다.

6
생성 제약

근거 기반 답변, 거절, 인용 형식, 안전 정책을 정의합니다.

7
회귀 평가

변경 전후를 같은 질문셋으로 비교하고 failure bucket을 관리합니다.

8
운영 관측

trace, 비용, 지연 시간, stale index, 사용자 피드백을 모니터링합니다.

8부. 마무리

TAKEAWAY

현업 RAG는 검색 + 생성이 아니라 데이터 + 검색 + 평가 + 운영이다.

NAIVE

빠른 시작

색인화, 검색, 생성의 기본 골격.

ADVANCED

품질 개선

검색 전·후 처리, rerank, 압축.

MODULAR

운영 확장

라우팅, 평가, 보안, 관측 가능성.

REFERENCES

참고문헌

주 참고문헌

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., Wang, H. (2024). *Retrieval-Augmented Generation for Large Language Models: A Survey.* arXiv:2312.10997.

발표에서 언급한 주요 기법

Lewis, P. et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.*

Gao, L. et al. (2022). *HyDE: Precise Zero-Shot Dense Retrieval without Relevance Labels.*

Ma, X. et al. (2023). *Query Rewriting for Retrieval-Augmented Large Language Models.*

Asai, A. et al. (2023). *Self-RAG.*

Jiang, Z. et al. (2023). *Active Retrieval Augmented Generation.*

평가와 운영 관련

Es, S. et al. (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation.*

Saad-Falcon, J. et al. (2023). *ARES: An Automated Evaluation Framework for RAG Systems.*

Chen, J. et al. (2023). *Benchmarking Large Language Models in Retrieval-Augmented Generation.*

Liu, N. F. et al. (2023). *Lost in the Middle: How Language Models Use Long Contexts.*

그림은 제공된 `images` 폴더의 RAG Survey 기반 도식 자료를 사용했습니다.