RAG 이모저모

시작(2020)부터 Long-Context 시대(2024)까지

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks
(Lewis et al., 2020)
+
RAG or Long-Context LLMs? A Comprehensive Study
(Li et al., Google DeepMind, 2024)

진동환

MOAI 세미나 · 2026

POSTECH

오늘의 발표 순서

01
RAG란 무엇인가?

기본 개념과 Motivation

02
클래식 RAG 아키텍처

DPR + BART, 두 모델 변형(2020)

03
실험 결과 & 레거시

오픈도메인 QA, Jeopardy, FEVER

04
RAG vs. Long-Context LLM

2024년의 새로운 대결

05
SELF-ROUTE 제안

비용과 성능의 최적 균형점

06
비판적 고찰 & 향후 연구 방향

한계점과 오픈 리서치 문제

PART 01

RAG란 무엇인가?

언어모델은 왜 외부 지식이 필요한가

LLM만으로는 부족한 이유 — 세 가지 근본 문제

GPT, BERT, T5 같은 pre-trained LLM은 파라미터 속에 지식을 저장합니다. 그런데 이 방식에는 구조적인 한계가 있습니다.

❄️ 지식의 동결 (Knowledge Staleness)

학습 완료 후 세계 지식이 바뀌어도 모델 파라미터는 고정
→ 업데이트하려면 수십억 파라미터 전체를 다시 학습해야 함
예: 2023년에 학습된 모델은 2024년 대통령을 모름

👻 환각 현상 (Hallucination)

지식이 파라미터에 분산 저장 → 출처 역추적 불가
→ 유창하지만 사실이 아닌 내용을 자신 있게 생성
예: T5-11B(110억 파라미터)도 NQ에서 34.5% EM

🔍 설명 불가능성 (No Provenance)

어떤 근거로 그 답을 냈는지 설명할 수 없음
→ 의료·법률 등 고위험 도메인에서 치명적 문제
→ 사용자 신뢰 확보 어려움

인간은 어떻게 행동하는가?

❄️ **지식의 동결 (Knowledge Staleness)**

학습을 통해 새로운 지식을 얻음

👻 **환각 현상 (Hallucination)**

사실이 아닌 내용을 자신 있게 말하다가도
팩폭 맞고 수정이 가능함

🔍 **설명 불가능성 (No Provenance)**

인터넷, 사전을 통해 출처를 가져옴

Google

NAVER



나무위키

해결책 : Parametric + Non-parametric Memory 결합

RAG는 두 가지 종류의 '기억 장치'를 하이브리드로 결합합니다.

Parametric Memory(미리 학습됨)

예: BART, T5, GPT

✗ 지식이 가중치(파라미터)에 내장

✗ 학습 이후 지식 고정

✗ 출처 추적 불가

✗ 업데이트 = 전체 재학습

✓ 유창한 언어 생성에 강점

+

Non-parametric Memory(외부 지식 창고)

예: Wikipedia Dense Index

지식이 외부 DB에 저장

✓ 인덱스 교체만으로 업데이트

✓ 검색 문서로 출처 제시 가능

✓ 해석 가능 (human-readable)

✓ 사람이 직접 편집 가능

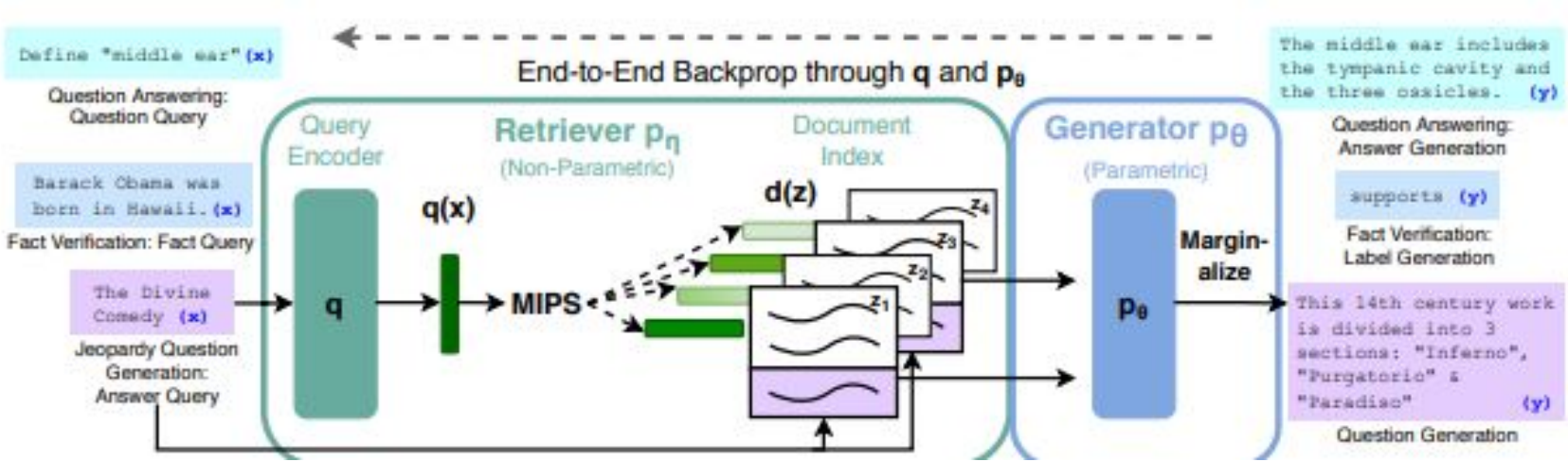
→ RAG: 두 메모리를 함께 사용하는 범용 seq2seq 프레임워크

PART 02

RAG 아키텍처 (2020)

DPR + BART, End-to-End 학습의 비밀

RAG 전체 아키텍처 한눈에 보기



핵심 컴포넌트 상세

Retriever (DPR)

bi-encoder: BERT 두 개로 query와 document를
각각 dense vector로 변환 → 내적으로 유사도 계산

MIPS(Maximum Inner Product Search)

Maximum Inner Product Search
FAISS로 2100만 문서 중 top-K를 sub-linear time에 검색

Generator (BART)

400M 파라미터 seq2seq transformer
입력: [query + 검색 문서] 를 단순 concatenation

End-to-End 학습

Retriever + Generator를 동시 fine-tuning
Document encoder는 비용 문제로 고정(frozen)

Retriever 심층 분석 : DPR (Dense Passage Retriever)

Bi-Encoder 구조

Query Encoder
BERT_q → q(x)

Doc Encoder
BERT_d → d(z)

$$p_\eta(z|x) \propto \exp\left(\mathbf{d}(z)^\top \mathbf{q}(x)\right)$$

→ 내적(Inner Product)이 클수록 관련 문서

MIPS (Maximum Inner Product Search)

FAISS + Hierarchical Navigable Small World 근사
→ 2100만 문서를 Sub-linear Time에 검색

Q. Document Encoder는 왜 고정(Frozen)했나?

문제

Doc encoder 업데이트 시
2100만 문서 전체를 매 step마다 재임베딩 필요

비용

GPU 연산 비용이 천문학적
REALM은 pre-training 중 이를 수행 → 극히 비쌈

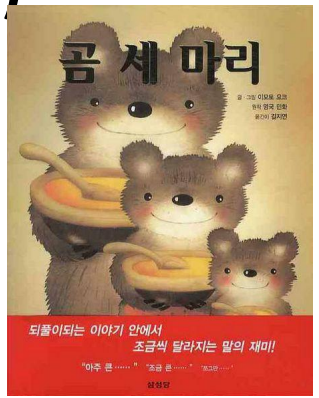
결론

Query encoder와 BART만 fine-tune해도
동일한 수준의 성능 달성 가능함을 실험으로
확인

Retriever 심층 분석 : DPR (Dense Passage Retriever)

Bi-Encoder 구조

Query Encoder
BERT_q → q(x)



21. 실수 전체의 집합에서 미분가능한 함수 $f(x)$가 다음 조건을 만족시킬 때, $f(-1)$의 값은? [4점]

(가) 모든 실수 x에 대하여
$2\{f(x)\}^2 f'(x) = \{f(2x+1)\}^2 f'(2x+1)$이다.

(나) $f\left(-\frac{1}{8}\right) = 1$, $f(6) = 2$

① $\frac{\sqrt[3]{3}}{6}$ ② $\frac{\sqrt[3]{3}}{3}$ ③ $\frac{\sqrt[3]{3}}{2}$ ④ $\frac{2\sqrt[3]{3}}{3}$ ⑤ $\frac{5\sqrt[3]{3}}{6}$

Doc Encoder
BERT_d → d(z)



RAG-Sequence vs. RAG-Token

RAG-Sequence
>한국어로 말할 때

RAG-Token
>영어로 말할 때

안녕하세요 제 이름은
진동환입니다.
....
저는 포스텍 24학번이고
AI스터디를 하고 있습니다.



Hello….My…name…i
s…Donghwan…I…a
m…a…student…in…
postech

두 모델 변형: RAG-Sequence vs. RAG-Token

핵심 차이: latent variable z(검색된 문서)를 언제 marginalize하는가?

RAG-Sequence

"하나의 문서로 전체 시퀀스 생성"

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x,z,y_{1:i-1})$$

- 문서 z를 고정하고 전체 시퀀스 y를 생성
- 각 문서별로 독립적인 Beam Search 수행
- 결과 hypothesis들을 모아 marginalize

✓ 강점: 맥락 일관성, 짧은 QA 답변에 최적

✓ NQ: 44.5 EM / TQA: 56.8 EM (SotA)

△ 한계: K번의 Beam Search로 연산 비용 높음

RAG-Sequence Decoder 구조

. 예시
질문: “모상우 교수님이 어느 학교의 교수인지 알려줘”
상위 3개 문서:
z_1: “모상우 교수는 POSTECH 산업경영공학과…” (0.8)
z_2: “모상우 박사는 KAIST 전기전자공학과…” (0.2)
각 문서의 beam search 결과를 결합하면,
최종 확률 계산 결과 **“POSTECH”** 이 가장 높은 확률로 선택됨.

RAG-Sequence Decoder 구조

구분	Thorough Decoding	Fast Decoding
행동 방식	"후보 KAIST야, 너 혹시 문서 A(포스텍내용) 에서도 나올 확률이 조금이라도 있니?" 하고 **전부 조사**	"후보 KAIST 너 문서 B에서만 나왔네? 그럼 문서 B 확률만 계산하고 끝!" 하고 **칼차단**
속도	느림 (모든 후보, 모든 문서 재평가)	**빠름 (구간별 계산 후 단순 합산)**
정확도	**높음 (수학적으로 완벽한 확률 계산)**	약간 낮음 (근사치 계산)

RAG-Sequence vs. RAG-Token

핵심 차이: latent variable z(검색된 문서)를 언제 marginalize하는가?

RAG-Token

"토큰마다 다른 문서 참조 가능"

$$p_{\text{RAG-Token}}(y|x) \approx \prod_{i}^{N} \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$$

- y_i 생성할 때마다 다른 z를 사용 가능
- Standard Beam Decoder에 직접 연결 가능
- 연산이 상대적으로 효율적

✓ 강점: 여러 문서의 정보를 한 답변에 합성

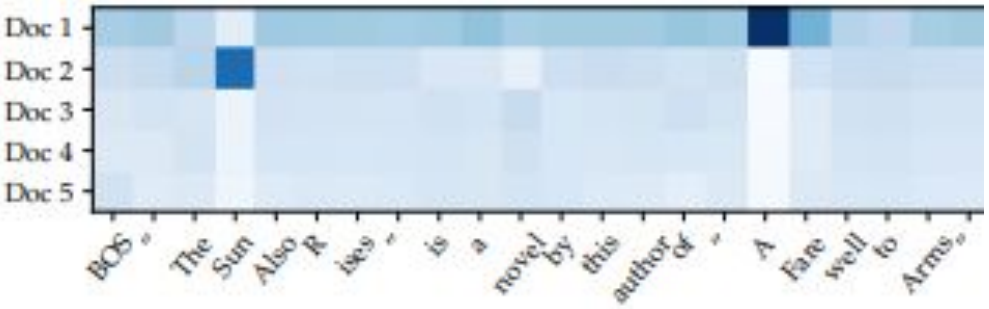
✓ Jeopardy Q-BLEU-1: 22.2 (SotA)

★ Hemingway 예시: 두 소설 제목을 한 문장에

RAG-Token Decoder 구조

Document 1: his works are considered classics of American literature ... His wartime experiences formed the basis for his novel **"A Farewell to Arms"** (1929) ...

Document 2: ... artists of the 1920s "Lost Generation" expatriate community. His debut novel, **"The Sun Also Rises"**, was published in 1926.



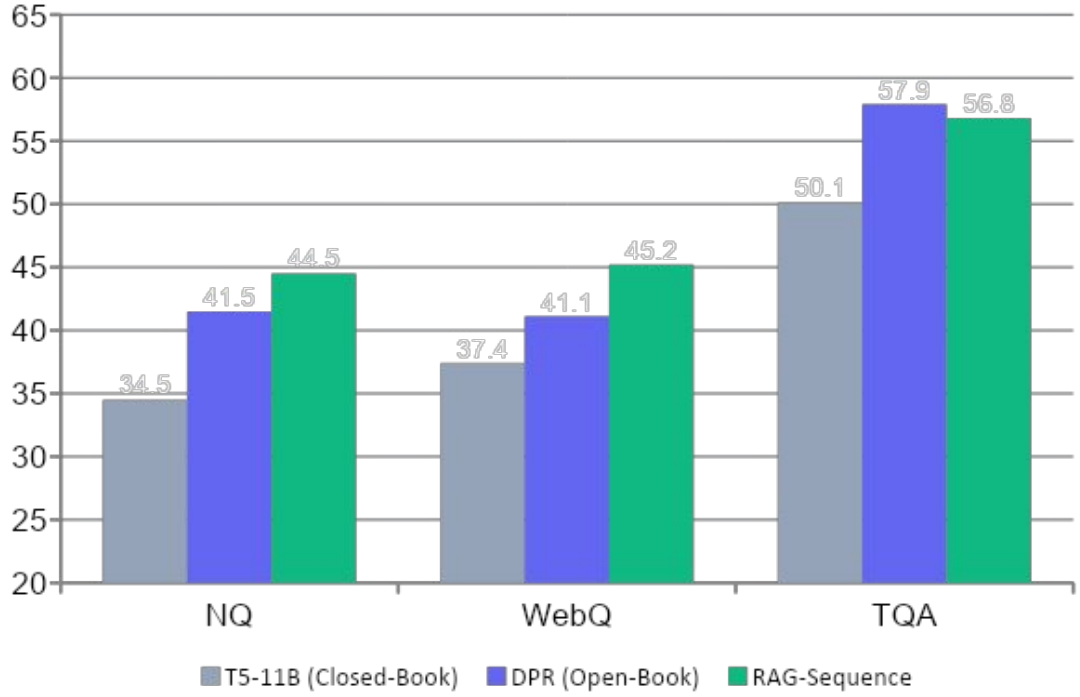
-> 모델의 시선(Attention)이 어디 머무는지를 실시간으로 보여주는 지도

PART 03

실험 결과 & RAG의 레거시

SotA 성능, Hallucination 감소, 그리고 현대 LLM에 미친 영향

Open-Domain QA 성능: 모든 태스크에서 State-of-the-Art



주요 발견

파라미터 효율성
626M param으로
T5-11B(11B) 능가
→ 18배 적은 파라미터

답이 없어도 생성
검색 문서에 정답이
없는 경우도 NQ에서
11.8% 정확도 달성

Index Hot-swap
인덱스 교체만으로
세계 지식 업데이트
(재학습 불필요)

Exact Match (EM) 점수 기준. 높을수록 좋음. TQA는 Wiki 테스트셋 기준.

Generation/Classification

Model	Jeopardy B-1	Jeopardy QB-1	MSMARCO R-L	MSMARCO B-1	FVR3 Label Acc.	FVR2 Label Acc.
SotA	-	-	**49.8***	**49.9***	**76.8**	**92.2***
BART	15.1	19.7	38.2	41.6	64.0	81.1
RAG-Tok.	**17.3**	**22.2**	40.1	41.5	72.5	89.5
RAG-Seq.	14.7	21.4	40.8	44.2		

Ex) Q: 모상우,
A:모상우 교수님은 KAIST에서 박사를 마치시고 POSTECH산업경영공학과 교수로 재직중이십니다

RAG 2020이 현대 LLM 생태계에 남긴 유산

이 논문이 발표된 2020년 이후, 현재까지 이어지는 기술적 기초를 마련했습니다.

1 Dense Retrieval 표준화

DPR + FAISS 조합이 수백 개의 후속 연구에서 기본 retrieval pipeline으로 채택됨. BM25(단어 겹침)보다 dense retrieval(의미 검색)이 대부분의 태스크에서 우월함을 증명.

2 End-to-End Trainable RAG 원형

Retriever와 Generator를 별개로 학습하지 않고 joint training하는 패러다임 정착. 이후 RLHF 기반 retrieval 최적화, feedback loop RAG 등으로 발전.

3 Knowledge Updatability 개념화

"Non-parametric memory는 교체 가능하다" → Temporal Knowledge Graph, Continual Learning과 결합된 다양한 연구로 이어짐. Hot-swap 실험으로 실증.

4 파라미터 효율성의 증명

626M param이 T5-11B(11B param)를 능가함을 보여줌. 모든 지식을 파라미터에 쑤셔넣을 필요가 없다는 패러다임 전환의 시작.

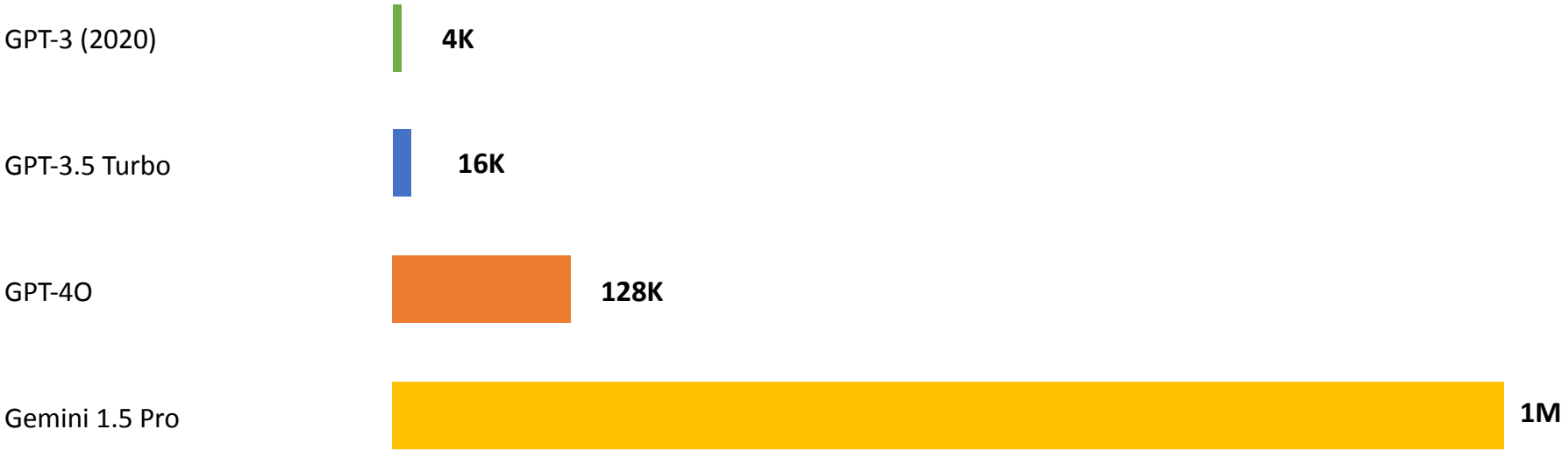
Discussion: 한계점

① Retrieval Quality의 병목
② Context Window의 제약
③ Retrieval Collapse 문제
④ 단방향 흐름의 한계.

패러다임의 충돌: Long-Context LLM의 등장

RAG의 전제: "LLM의 context window가 짧다" — 그런데 이 전제가 2024년에 흔들리기 시작했습니다.

Context Window 크기 비교 (상대적)



핵심 질문: 이제 문서 전체를 LLM에 직접 넣어버릴 수 있다면, RAG는 여전히 필요한가?

PART 04

RAG vs. Long-Context LLM (2024)

Gemini 1.5와 GPT-4O의 등장이 RAG의 입지를 바꾸다

실험 설계: 공정한 비교를 위한 준비

9개 데이터셋, 3개 최신 LLM, 2개 Retriever를 사용한 종합 벤치마크

평가 데이터셋 (9개)

LongBench (7개, 평균 7k 단어)

- NarrativeQA (소설·영화 QA)
- Qasper (NLP 논문 QA)
- HotpotQA (2-hop 추론)
- 2WikiMultihopQA (5-hop까지)
- MuSiQue (4-hop 다중 추론)
- MultiFieldQA, QMSum

∞Bench (2개, 평균 147k 단어)

- En.QA, En.MC

평가 LLM (3개)

Gemini-1.5-Pro

→ 최대 1M 토큰 (Google)

GPT-4O

→ 최대 128K 토큰 (OpenAI)

GPT-3.5-Turbo

→ 최대 16K 토큰 (OpenAI)

RAG 설정: 청크 300단어

Top-K = 5 (기본값)

Retriever (2개)

Contriever (기본)

→ 대조 학습 기반 dense retriever

→ BM25보다 BEIR에서 우수

Dragon

→ 일반화된 dense retriever

→ supervised/zero-shot 모두 강함

→ 결과: 두 retriever 모두 동일 경향

실험 설계: RAG와 LC 작동의 차이

RAG · 청크 몇 개만 읽기



- **비유**: 위키 색인 보고 **관련 문단 몇 개**만 펼쳐 읽기
- **동작**: 질의 → Retriever가 top-k 청크 선택 → LLM은 그 청크만 보고 답
- **입력**: 청크 ~500토큰 × k개 (보통 수천 토큰)
- **장점**: 저비용·저지연, 대규모 코퍼스 확장 가능
- **약점**: 검색이 틀리면 정답 근거 자체가 누락

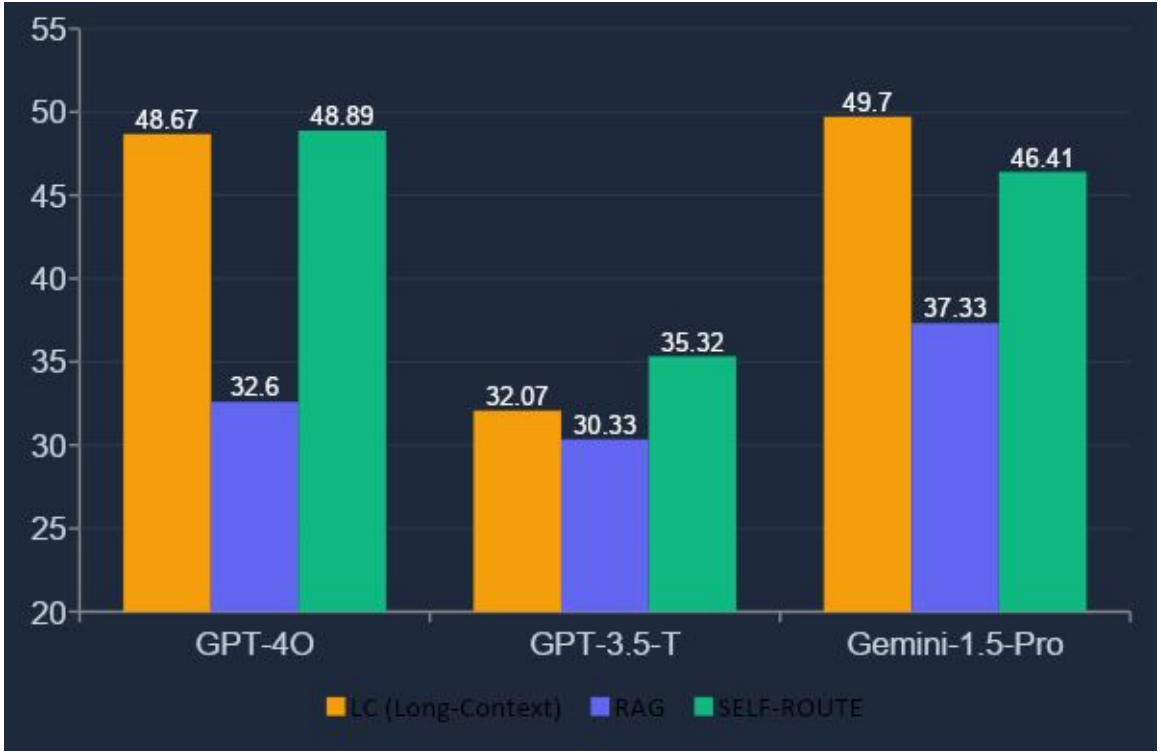
LC · 위키피디아 전체 읽기



- **비유**: 위키피디아 **한 권을 통째로** 펼쳐서 읽기
- **동작**: 검색 없이 문서 전체를 컨텍스트로 넣고 LLM이 직접 답
- **입력**: 문서 전체 (수만~수십만 토큰)
- **장점**: 검색 실패가 원천적으로 없음, 근거 모두 노출
- **약점**: 고비용·고지연, 긴 컨텍스트 내 길찾기 (lost-in-the-middle)

핵심 차이: RAG는 **"무엇을 읽을지" 검색이 선택**, LC는 **"전부 읽고" 모델이 알아서 찾음**

핵심 실험 결과: 충분한 리소스에서 LC가 RAG를 능가한다



LC vs. RAG 성능 격차

GPT-4O	**+13.1%p**
Gemini-1.5-Pro	**+7.6%p**
GPT-3.5-Turbo	**+3.6%p**

예외 케이스

∞Bench (147k 단어) + GPT-3.5-T (16K window)에서는 RAG가 오히려 승리!
→ 컨텍스트가 window를 훨씬 초과할 때 RAG 효과적

RAG는 언제, 왜 실패하는가? — 4가지 원인 분석

저자들은 RAG가 "unanswerable"로 판정하는 실패 사례를 분석해 4가지 패턴을 도출했습니다.

A Multi-step Reasoning 필요

"song XXX를 부른 가수의 국적은?" → 2단계 추론 필요
첫 검색 결과가 두 번째 검색 쿼리 생성에 필요한데,
RAG는 단일 패스 검색이라 처리 불가

HotpotQA, 2WikiMQA, MuSiQue에서 지배적

B General Query (모호한 쿼리)

"그 그룹은 Dave의 탈퇴에 대해 어떻게 생각했나?"
→ 쿼리가 너무 포괄적 → retriever가 적절한
임베딩 벡터 형성 불가

QMSum (회의록 요약) 데이터셋에서 많음

C 복잡하고 긴 쿼리

쿼리 자체가 너무 길고 복잡해서
Dense Retriever의 인코딩 품질 저하
→ 어떻게 보면 이건 LLM의 장점

모든 데이터셋에서 산발적으로 발생

D Implicit Query (전체 이해 필요)

"우주선 뒤 그림자의 원인은?" → 긴 서사 전체를
읽고 암묵적으로 연결해야만 답 가능
어떤 검색 전략으로도 해결 어려움

NarrativeQA (소설·영화) 에서 지배적

PART 05

SELF-ROUTE: 최적의 균형점

Model Self-reflection으로 RAG와 LC를 동적 라우팅

LC만으로는 아쉬운 이유 – 비용 문제

핵심 아이디어: RAG와 LC가 63%의 쿼리에서 동일한 예측을 낸다 → 어차피 같다면 저렴한 RAG를 쓰면 됨!

TASK: 졸업 논문 쓰기

연구실 청소해주기	?
논문 써주기	?





SELF-ROUTE 작동 원리: 2단계 라우팅

STEP 1 · RAG-and-Route 비용: ~1,500 tokens

1. Query에 대해 Top-K 청크 검색 (RAG)
2. LLM에 청크 + 쿼리 제공 (~1.5K tokens)
3. 두 가지 동시 수행:
   ① 청크만으로 답변 가능한지 판단
   ② 가능하면 즉시 RAG 답변 생성

핵심 프롬프트:

"Write unanswerable if the query
can not be answered based on
the provided text"

STEP 2 · Long-Context (필요시만) 비용: ~100K tokens

Step 1에서 "unanswerable" 판정 시만 실행

전체 문서 컨텍스트를 LLM에 제공
→ LC 방식으로 최종 답변 생성

전체 쿼리의 약 18~43%만 이 단계 진입
(모델별 차이 있음)

이 단계가 적을수록 비용 절감!

STEP1에서 판단 잘못하는거 아님?

결론: STEP1(Retriever)만의 문제로 보기 어렵다

- **근거 1 — Long-Context(LC) 비교 실험**
  - 검색 단계를 우회하고 문서 전체를 컨텍스트로 넣어도(LC), RAG가 틀리는 문제는 LC도 비슷하게 틀렸음
  - 즉, 정답 근거가 컨텍스트에 **이미 들어있어도** 생성(STEP2)이 답을 못 만들어내는 경우가 다수
- **근거 2 — 검색 품질과 최종 정답률의 약한 상관**
  - Recall@k가 올라가도 EM/F1이 비례해서 오르지 않는 구간이 존재 → 병목은 reasoning·generation 쪽
- **시사점**
  - STEP1 개선(좋은 임베딩·리랭커)만으로는 한계 → STEP2의 **근거 활용 능력**(grounding, 추론, instruction following)도 함께 개선해야 함

SELF-ROUTE 성능: LC에 준하는 성능을 38% 비용으로

46.41

SELF-ROUTE
(Gemini-1.5-Pro)

LC: 49.70 (-2.2%p)

48.89

SELF-ROUTE
(GPT-4O)

LC: 48.67 (오히려 +0.2%p)

38.4%

토큰 사용량
(Gemini 기준)

LC 대비 61.6% 비용 절감

모델별 상세 결과 (Contriever retriever 기준)

모델	LC 성능	RAG 성능	SELF-ROUTE	토큰 사용 %
Gemini-1.5-Pro	**49.70**	37.33	**46.41**	**38.4%**
GPT-4O	**48.67**	32.60	**48.89**	**61.4%**
GPT-3.5-Turbo	**32.07**	30.33	**35.32 (+1.7%p!)**	**38.9%**

PART 06

비판적 고찰 & 향후 연구

이 논문의 한계와 미래 연구 방향

비판적 고찰 (1): 데이터셋의 구조적 한계

Distribution Mismatch

심각도 : 높음

LongBench 평균 7k 단어, ∞Bench 100k — 실제 enterprise 환경(법률 수백 페이지, 코드베이스 등)의 worst-case를 대표하지 않음. 논문의 결론이 실제 production에서 그대로 적용될지 불확실.

Synthetic Dataset Artifact

심각도 : 높음

PassKey 태스크에서 쿼리 문구만 바꿔도 RAG 정확도가 80% →4.6%로 폭락. 이는 벤치마크가 retrieval 능력보다 특정 패턴 매칭을 테스트하고 있을 수 있음을 시사. 결과 해석에 주의 필요.

Data Leakage 미해결

심각도 : 중간

"based only on the provided passage" 프롬프트로 억제 시도했지만 완전한 격리 불가. GPT-4O, Gemini가 Wikipedia 기반 데이터를 이미 대량 학습했을 가능성. LC의 우위가 '긴 컨텍스트 이해'인지 '파라미터 기억'인지 분리하기 어려움.

Query-only RAG 가정

심각도 : 중간

이 논문의 RAG는 단일 쿼리 → 검색 → 생성의 단순 파이프라인. Iterative RAG, Chain-of-Thought + RAG, 쿼리 재작성(Query Rewriting) 등 더 발전된 RAG 방식은 비교 대상에 포함되지 않음.

비판적 고찰 (2): SELF-ROUTE 상용화 시 잠재 리스크

⚡ Latency 문제

최악의 경우(unanswerable 판정) RAG 패스 + LC 패스 두 번 LLM 호출 발생. LC 호출은 수만~수십만 토큰 → 수십 초 지연 가능. 실시간 챗봇·고객 지원처럼 latency SLA 엄격한 서비스에서는 치명적.

⚖️ Self-reflection 불균형

Answerable 판정률이 GPT-4O(57%)와 Gemini(77%)에서 크게 다름. 이는 모델별 alignment 차이에서 비롯된 것 → 특정 모델에 최적화한 임계값이 다른 모델에서는 suboptimal. 범용 배포 어려움.

❌ False Negative/Positive 제어 불가

RAG가 가능한 쿼리를 unanswerable로 오판 → 불필요한 LC 호출(비용 증가). RAG 불가 쿼리를 answerable로 오판 → 틀린 답변이 최종 출력. 이 두 오류를 독립적으로 제어할 메커니즘 없음.

💰 운영 비용 과소평가

논문은 LLM API 과금(토큰 수)만 비교. 실제로는 retrieval 인덱스 구축·유지 비용, 벡터 DB 운영 비용, 청크 업데이트 비용이 추가로 발생. 동적으로 변하는 지식 베이스 유지 시 overhead 상당.

핵심 Takeaway & 향후 연구 방향

한 문장 요약: LC가 더 좋지만 비용이 크다 → SELF-ROUTE로 97% 성능을 38% 비용에 달성하되, Latency와 Calibration 설계가 전제되어야 한다.

어떤 방법을 선택할까? — 의사결정 가이드라인

If Context > Model Window를 크게 초과 → **RAG**

If 최신 강력 모델 + 비용 무제한 → **LC**

If 비용 제약 + 성능도 중요 → **SELF-ROUTE**

If Multi-hop reasoning 중심 태스크 → **Iterative RAG 연구 필요**

미래 연구 방향

- RAG + RLHF: 사용자 피드백으로 retrieval 품질을 직접 최적화
- Iterative / Multi-hop RAG: 단일 패스 검색의 한계 극복
- Chain-of-Thought + RAG: 추론 중 필요한 시점에 추가 검색 (FLARE 등)
- Adaptive RAG: 쿼리 복잡도를 예측하여 검색 전략 자동 선택

감사합니다

Q & A

참고 논문

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

Li et al. (2024). Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study. arXiv:2407.16833.

APPENDIX

Table 1: Open-Domain QA Test Scores. For TQA, left column uses the standard test set for Open-Domain QA, right column uses the TQA-Wiki test set. See Appendix D for further details.

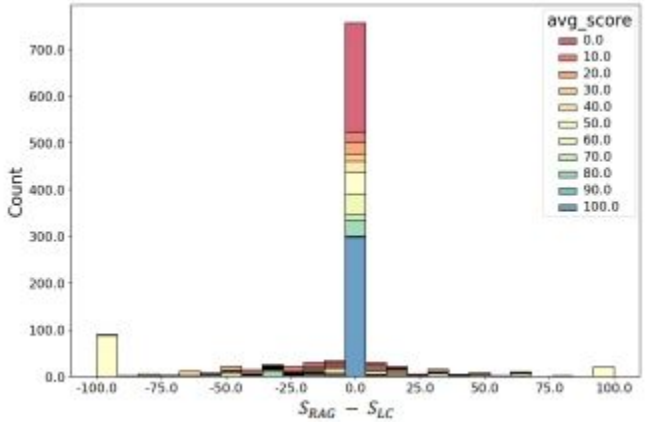
	Model	NQ	TQA	WQ	CT
Closed Book	T5-11B [52]	34.5	- /50.1	37.4	-
	T5-11B+SSM[52]	36.6	- /60.5	44.7	-
Open Book	REALM [20]	40.4	- / -	40.7	46.8
	DPR [26]	41.5	**57.9**/ -	41.1	50.6
	RAG-Token	44.1	55.2/66.1	**45.5**	50.0
	RAG-Seq.	**44.5**	56.8/**68.0**	45.2	**52.2**

Table 2: Generation and classification Test Scores. MS-MARCO SotA is [4], FEVER-3 is [68] and FEVER-2 is [57] *Uses gold context/evidence. Best model without gold access underlined.

Model	Jeopardy B-1	Jeopardy QB-1	MSMARCO R-L	MSMARCO B-1	FVR3 Label Acc.	FVR2 Label Acc.
SotA	-	-	**49.8***	**49.9***	**76.8**	**92.2***
BART	15.1	19.7	38.2	41.6	64.0	81.1
RAG-Tok.	**17.3**	**22.2**	40.1	41.5	72.5	89.5
RAG-Seq.	14.7	21.4	40.8	44.2		

Table 3: Examples from generation tasks. RAG models generate more specific and factually accurate responses. '?' indicates factually incorrect responses, * indicates partially correct responses.

Task	Input	Model	Generation
MS-MARCO	define middle ear	BART	?The middle ear is the part of the ear between the middle ear and the nose.
		RAG-T	The middle ear is the portion of the ear internal to the eardrum.
		RAG-S	The middle ear includes the tympanic cavity and the three ossicles.
	what currency needed in scotland	BART	The currency needed in Scotland is Pound sterling.
		RAG-T	Pound is the currency needed in Scotland.
		RAG-S	The currency needed in Scotland is the pound sterling.
Jeopardy Question Gener-ation	Washington	BART	?This state has the largest number of counties in the U.S.
		RAG-T	It's the only U.S. state named for a U.S. president
		RAG-S	It's the state where you'll find Mount Rainier National Park
	The Divine Comedy	BART	*This epic poem by Dante is divided into 3 parts: the Inferno, the Purgatorio & the Purgatorio
		RAG-T	Dante's "Inferno" is the first part of this epic poem
		RAG-S	This 14th century work is divided into 3 sections: "Inferno", "Purgatorio" & "Paradiso"



			Avg	Narr	Qasp	Mult	Hotp	2Wiki	Musi	Sum	En.QA	En.MC
Gemini-1.5-Pro	1-1	LC	**49.70**	**32.76**	**47.83**	**52.33**	**61.85**	**62.96**	40.22	**20.73**	**43.08**	**85.57**
	1-2	RAG	37.33	22.54	44.68	49.53	48.36	54.24	26.56	19.51	19.46	51.09
	1-3	SELF-ROUTE	46.41	28.32	45.23	51.47	55.18	62.68	**40.66**	19.77	37.51	76.86
	1-4	answerable %	76.78	73.00	85.00	96.67	84.50	81.00	58.50	93.50	56.41	62.45
	1-5	token %	38.39	23.07	49.93	36.88	32.97	53.49	56.14	17.96	42.25	32.84
GPT-4O	2-1	LC	48.67	**32.78**	44.54	**55.28**	**62.42**	**70.69**	41.65	**21.92**	32.36	76.42
	2-2	RAG	32.60	18.05	46.02	50.74	36.86	50.21	16.09	19.97	14.43	41.05
	2-3	SELF-ROUTE	**48.89**	31.36	**47.99**	53.17	62.14	70.14	**41.69**	21.31	**34.95**	**77.29**
	2-4	answerable %	57.36	44.00	67.50	94.00	52.50	62.00	30.00	92.00	27.07	47.16
	2-5	token %	61.40	66.40	72.25	39.65	65.79	77.05	85.00	20.26	73.01	53.21
GPT-3.5-Turbo	3-1	LC	32.07	23.34	**42.96**	49.19	45.33	41.04	17.92	19.61	14.73	34.50
	3-2	RAG	30.33	18.22	38.15	49.21	37.84	35.16	16.41	18.94	15.39	43.67
	3-3	SELF-ROUTE	**35.32**	**24.06**	38.65	**52.07**	**47.28**	**44.62**	**34.44**	**19.88**	**22.03**	**44.54**
	3-4	answerable %	74.10	71.50	80.00	91.33	68.50	69.00	47.00	93.50	50.43	95.63
	3-5	token %	38.85	20.56	55.08	35.29	48.70	65.91	65.08	16.40	38.17	4.50

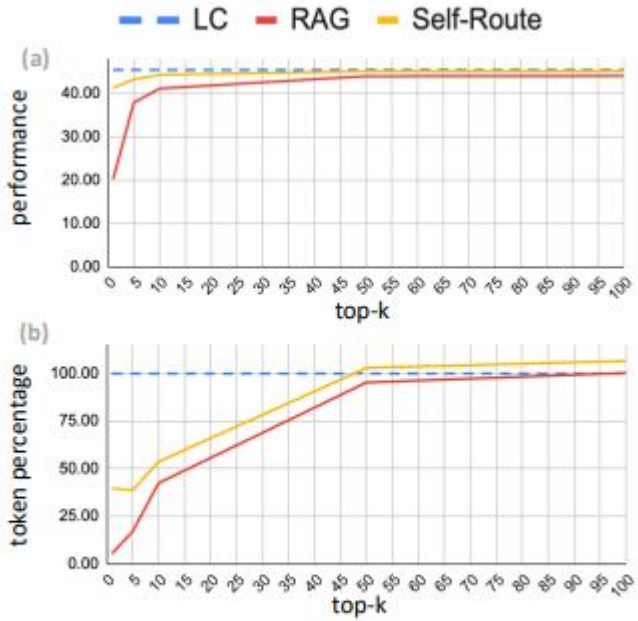


Figure 3: Trade-off curves between (a) model performance and (b) token percentage as a function of k.