Query-Aware Context Compression

26.06.11 (목) / 홍성태
ghdchlwls123@korea.ac.kr

MAY 14, 2026

Query-Aware Context Compression for Better Snippets

Improving the quality-efficiency frontier of model context through query-aware context compression models.

LooComp: Leverage Leave-One-Out Strategy to Encoder-only Transformer for Efficient Query-aware Context Compression

Thao Do, Dinh Phu Tran, An Vo, Seon Kwon Kim, Daeyoung Kim
Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, South Korea
{thaodo, phutx2000, an.vo, lukaskim, kimd}@kaist.ac.kr

MAY 14, 2026

Query-Aware Context Compression for Better Snippets

Improving the quality-efficiency frontier of model context through query-aware context compression models.

Query-Aware Context Compression for Better Snippets

검색/RAG 시스템에서 더 좋은 snippet을 만드는 방법

- 검색 기반 AI 시스템의 답변은 결국 evidence에서 시작.
- 사용자 질문에 답하기 위해 시스템은 웹 문서, 제목, 요약, chunk, snippet 등을 검색, 이를 model이나 agent에 context로 전달.
- 문제는 raw retrieved context는 항상 깨끗하지 않으며, 문서 안에는 사용자의 질문에 필요한 핵심 evidence가 포함되어 있을 수 있지만, 동시에 navigation text, 광고, metadata, boilerplate, off-topic content, 중복 정보도 함께 포함.

LLM에게 더 많은 context를 주는 것이 항상 좋은 것은 아니다.

- 중요한 것은 context의 양이 아니라, 질문에 실제로 필요한 high-signal evidence를 얼마나 정확하게 전달하느냐임.
- Perplexity는 이를 snippet generation을 일반적인 요약 문제가 아니라, query-aware context compression 문제로 정의.
- 즉, 각 query와 candidate result에 대해 질문에 필요한 span은 보존하고, 나머지 distractor는 제거하는 방식이다.

왜 Raw Context가 문제인가?

Noisy context는 accuracy, latency, cost를 동시에 악화시킴

Raw retrieved context에는 답변에 필요한 정보뿐 아니라, 실제로 사용할 필요가 없는 다양한 distractor가 존재함..

첫째, accuracy를 떨어뜨린다. 강력한 LLM이라도 capacity는 무한하지 않으며, 긴 irrelevant context가 주어지면 모델은 실제 사용자 요청과 관련 없는 정보를 처리하는 데 capacity를 낭비함. ➔ 핵심 fact를 놓치거나, 낮은 품질의 정보를 과대평가하거나, source에 제대로 grounded되지 않은 주장을 만들 가능성이 높아진다. 이를 “context rot”이라고 함.

둘째, latency를 증가시킨다. 긴 context는 downstream model이 처리해야 하는 token 수를 늘림. 특히 query 하나에 여러 candidate page의 evidence가 필요한 경우, prefill 비용뿐 아니라 reasoning 과정에서의 비용도 커짐. irrelevant context가 많을수록 모델은 어떤 정보가 중요한지 판단하기 위해 더 많은 reasoning token을 사용.

셋째, serving cost를 높인다. 불필요한 token도 모두 input token으로 계산되며, noisy context는 reasoning token까지 증가. 즉, distractor는 품질을 낮출 뿐 아니라 더 나쁜 품질을 위해 더 많은 비용을 지불. 따라서 좋은 검색 시스템은 단순히 관련 문서를 많이 가져오는 것에서 끝나면 안 된다. Answer model에 전달되기 전에 context를 정제하고, 질문에 필요한 evidence만 남겨야함.

Snippet Generation Reframing (1)

Snippet은 요약문이 아니라 evidence layer다

일반적으로 snippet을 만드는 방법은 크게 두 가지로 나눔.

1. selection-based method.

문서 안에서 관련 있어 보이는 window, chunk, sentence, passage를 선택해서 downstream model에 전달한다. 이 방식은 빠르고 원문을 유지할 수 있지만, 선택 단위가 너무 coarse하면 질문에 필요한 세부 evidence만 남기기 어려움.

2. generation-based method.

모델에게 문서를 읽게 한 뒤, user query에 맞는 summary를 새롭게 생성하게 함. 이 방식은 유연하지만 evidence layer에는 적합하지 않음. 생성 요약은 원문을 paraphrase할 수 있고, source에 없던 wording을 추가할 수 있으며, citation alignment를 어렵게 만든다. 또한 candidate page마다 요약을 생성해야 하므로 latency와 cost가 크게 증가함.

Snippet Generation Reframing (2)

그래서?

Perplexity는 이러한 이유로 generation-based summarization을 배제함.

snippet은 사람이 읽기 좋은 요약문이 아니라, answer model이 근거로 사용할 source-preserving evidence임

Query-aware extractive compression

query와 candidate result가 주어졌을 때, 해당 query를 지원하는 span은 보존하고 나머지는 제거.

이 방식은 세 가지 장점을 가짐.

- 원문 wording을 유지하므로 citation fidelity가 높다.
- 불필요한 context를 줄여 latency와 cost를 낮춘다.
- downstream model이 더 정확한 evidence에 집중하게 만든다.

Query-Aware Extractive Compression의 핵심 아이디어

같은 문서라도 질문에 따라 남겨야 할 span은 달라진다

- Query-aware context compression의 핵심은 문서 전체의 relevance를 판단하는 것이 아님.
- 중요한 것은 특정 query에 대해 문서 안의 어떤 span이 실제 evidence로 유용한지를 판단하는 것.
- 모델은 user query와 candidate web page의 title, summary, context text를 함께 입력받고, 문서 안의 span을 keep/drop으로 구분함. 사용자 질문에 답하는 데 필요한 evidence는 keep, 질문과 무관하거나 방해가 되는 정보는 drop.
- 이 방식의 핵심은 query-aware selection임.
    - 같은 문서라도 질문이 달라지면 남겨야 할 정보도 달라짐.
    - 따라서 compressor는 retriever와 역할이 다르며, Retriever가 관련 가능성이 있는 문서를 찾는다면, compressor는 그 문서 안에서 현재 query에 필요한 evidence 위치를 찾는 precision stage에 가까움.

Model Architecture

Bidirectional Encoder 기반 Context Compressor

- Perplexity는 context compression model의 backbone으로 pplx-diffusion 계열 bidirectional encoder를 사용함.
- encoder는 query와 candidate context를 함께 입력받아, 각 token이 전체 query와 문서 맥락을 모두 참고한 representation을 만들도록 구성.
- 기존 backbone은 context를 이해하는 역할을 하지만, 어떤 span을 남길지 직접 결정하지는 않음. 따라서 그 위에 lightweight compression head를 추가해 각 token/span의 keep score를 예측함.
- **Serving 단계에서는 token-level prediction을 sentence-level로 모으고, threshold와 token budget을 적용해 최종 snippet을 구성함.**

즉, 이 모델은 답변을 생성하는 모델이 아니라, answer model 이전에 필요한 evidence만 남기는 context 정제 모델임.

Supervision Pipeline

LLM-as-a-Judge 기반 span labeling

- Context compression을 학습하려면 document-level relevance label만으로는 부족함.
- 모델은 문서 전체의 관련성보다, 문서 안에서 어떤 span을 keep/drop해야 하는지를 배워야 함.
- Perplexity는 이를 위해 LLM-as-a-Judge 기반 span labeling pipeline을 사용함.

  : Judge model은 먼저 user query의 intent를 분석하고, 그 intent를 기준으로 candidate context에서 필요한 evidence span을 원문 그대로 추출함. 이후 추출된 span을 source text에 align하고, token-level keep/drop label로 변환함.
- Naive string matching으로 98%의 span을 복구했으며, regex-based matching을 추가해 recovery를 100%까지 높임.

최종적으로 약 750k query-document pair를 라벨링하고, 이를 사용해 compression head를 binary cross-entropy loss로 학습.

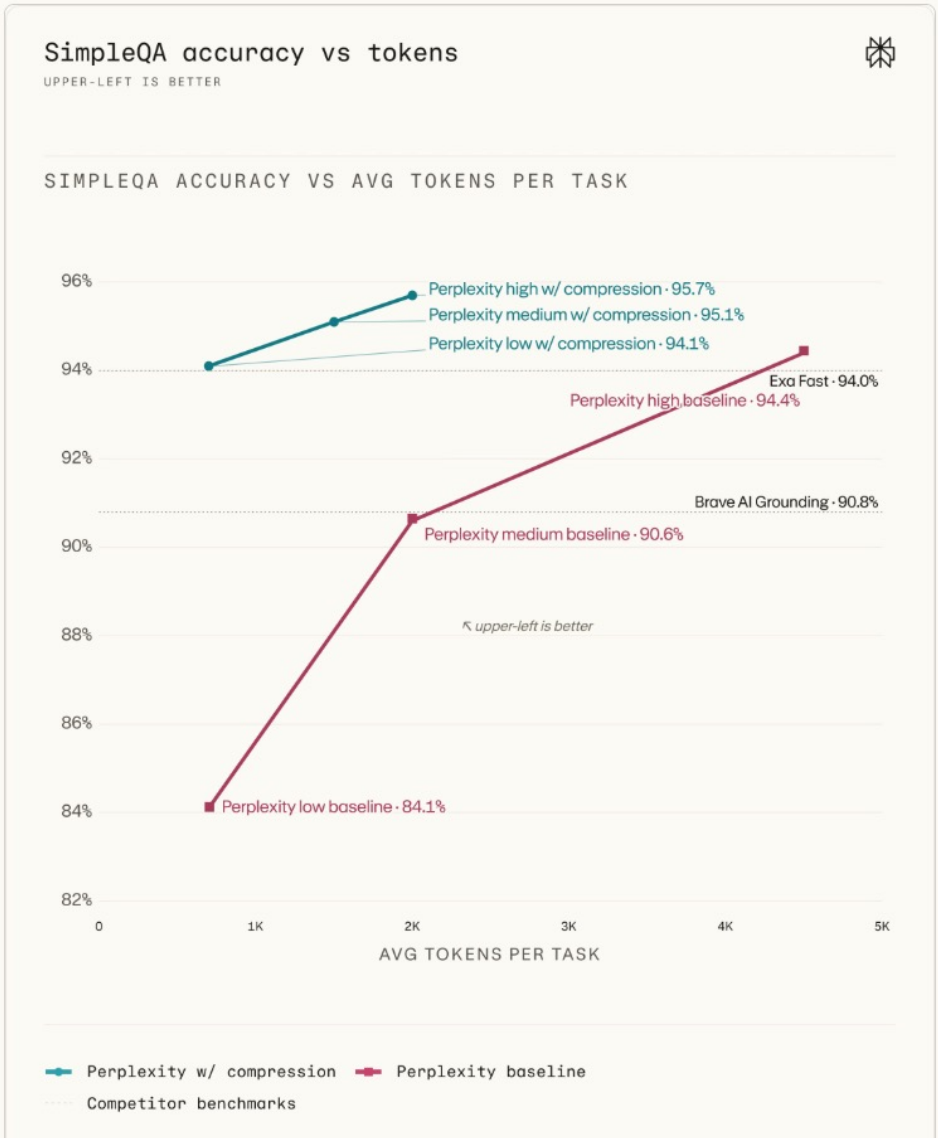
Production Latency Optimization

Compressor는 빠르지 않으면 의미가 없음

- Context compression은 answer model이나 agent에 context를 넘기기 전에 실행됨. 즉, downstream model 이전의 blocking operation. Compression으로 downstream token을 줄이더라도 compressor 자체가 너무 느리면 전체 시스템 관점의 이득이 사라짐.

- 또한 compression은 query-dependent임. 미리 문서만 압축해두는 방식이 아니라, query와 candidate document pair를 매번 함께 scoring해야 함. 검색 결과가 여러 개라면 각 candidate result마다 compressor 실행 필요.

Results



SimpleQA는 4,326개의 factual question으로 구성된 single-step factuality benchmark.

- Compression은 SimpleQA에서도 모든 preset에서 accuracy와 efficiency를 동시에 개선함.
- 대표적으로 medium preset with compression은 95% accuracy를 달성했고, 평균 200 tokens per document만 사용함.
- 평균 document length가 10,000 tokens 이상이라는 점을 고려하면 50배 이상의 compression ratio.
- 이 결과는 query-aware compression이 multi-step search뿐 아니라 단일 factual lookup에서도 효과적임을 보여줌.

Token Composition

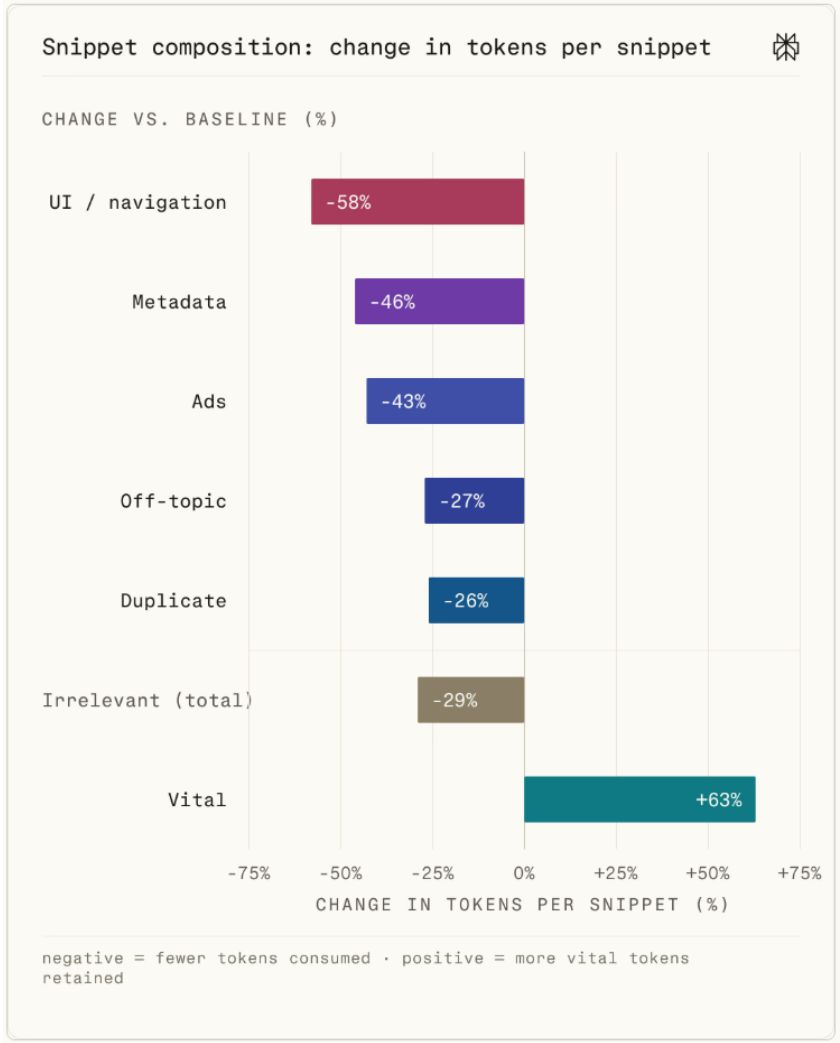


Figure 5. Change in per-snippet token composition from compression, averaged across 1,000 queries. The proportion of vital content increases, with corresponding decreases in non-vital content categories.

Compression은 context의 양이 아니라 정보 밀도 개선

- End-to-end benchmark 결과만으로는 compression이 왜 성능을 개선했는지 알기 어려움.
- Perplexity는 retrieved snippet의 각 token을 LLM judge로 분류해, snippet 내부 구성이 어떻게 바뀌는지 분석함.
- Token category는 여섯 가지. Vital은 사용자 질문에 답하는 데 필수적인 evidence, off-topic은 다른 목적에는 유용할 수 있지만 현재 query와 무관한 정보, ads는 광고성 distractor, metadata는 문서 관련 부가 정보, UI & navigation은 frontend 표시나 navigation 요소, duplicate는 반복 정보.

즉, compression은 snippet 안에서 useful evidence의 비율을 높이고, 모델을 방해하는 non-vital content를 제거함. Context를 단순히 짧게 만드는 것이 아니라, signal-to-noise ratio를 높이는 기술에 가까움.

Takeaways

- 검색/RAG 시스템에서 더 많은 context가 항상 더 좋은 답변을 의미하지는 않음.
- Raw context가 noisy하면 accuracy, latency, cost가 모두 악화될 수 있음.
- Snippet은 요약문이 아니라 evidence layer에 가까움. 따라서 source wording을 유지하는 extractive 방식이 citation fidelity와 grounding 측면에서 유리함.
- Compression은 query-aware해야 함. 같은 문서라도 사용자 질문에 따라 필요한 span은 달라짐.
- Production system에서는 quality뿐 아니라 latency도 중요함.

향후 RAG/search stack에서는 context curation이 핵심 모듈로 자리 잡을 가능성이 높다고 함.

Retriever → Reranker → **Query-Aware Compressor** → Generator

좋은 RAG 시스템은 더 많은 문서를 넣는 시스템이 아니라, 질문에 필요한 evidence를 가장 높은 밀도로 전달하는 시스템

LooComp: Leverage Leave-One-Out Strategy to Encoder-only Transformer for Efficient Query-aware Context Compression

Thao Do, Dinh Phu Tran, An Vo, Seon Kwon Kim, Daeyoung Kim
Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141,
South Korea

{thaodo, phutx2000, an.vo, lukaskim, kimd}@kaist.ac.kr

Introduction

Background

- 외부 지식을 활용해 LLM의 사실적 정확도를 높이고 Hallucination을 완화하는 핵심 기술로 자리 잡음.
- 복잡한 질문에 답하기 위해 검색 문서의 양을 늘리면 정보의 포괄성은 높아지지만, 시스템의 효율성이 저하.

Problem Statement

1. 긴 문맥의 부작용: 불필요한 정보(Noise) 유입으로 인한 성능 저하 및 막대한 연산 오버헤드 발생.
2. 기존 문맥 압축(Context Compression) 기술의 한계:

   Abstractive: 토큰 생성 과정에서 Latency 문제 / Extractive: 경직된 선택 기준으로 인해 복잡한 질문에 적응하지 못함.
3. 최근 연구(EXIT, Provence 등)의 비효율성:

   - 단순한 관련성 분류 작업에 무거운 디코더 기반(Decoder-based) LLM을 사용하여 연산 낭비 초래.
   - 문장의 맥락을 무시한 토큰 단위 가지치기(Pruning)로 인해 노이즈 발생 및 정확도 저하.

Introduction

LooComp

- LOO-Δ (Leave-One-Out): 특정 문장을 뺐을 때 정답 도출 확률이 얼마나 떨어지는지를 측정하여 문장의 '한계 기여도' 산출.
- 경량 인코더 모델 활용: ModernBERT 학습을 통해 매우 긴 문맥도 병렬로 빠르게 계산.
- Adaptive Gap-based Selection: 점수 분포에서 Gap을 찾아 질문 난이도에 맞춰 유동적으로 핵심 문장만 추출.

Contributions

- LOO-Δ 방식과 경량 모델(Encoder-only)을 결합하여, 메모리 요구량은 줄이고 병렬 연산을 통해 처리 속도를 극대화.
- 고정된 기준이 아닌 '적응형 격차 기반 선택'을 통해 압축률과 정보의 질을 동시에 보존.
- 5개의 주요 QA 벤치마크 평가 결과, 기존 베이스라인 대비 답변의 정확도 향상, 문맥 길이를 줄이고 압축 속도를 획기적으로 개선.

LooComp

Loo - Δ (Leave-One-Out Delta)의 개념

- 특정 문장을 문서 전체에서 제외했을 때, 그 문서가 Query의 정답을 도출하는 데 얼마나 기여하는지 그 하락폭(Drop)을 측정.
- 즉, "이 문장이 없어지면 정답을 찾기 얼마나 힘들어질까?"를 정량화.

Equation

- Δ_i 의 (문장 i의 중요도) 산출 방식
- 전체 문서를 D라 하고, 질문에 대한 단서 풍부도를 예측하는 모델 함수를 $f(q, D)$라 할 때:

$$\Delta_i = f(q, D) - f(q, D \setminus \{s_i\})$$

- $f(q, D)$: 전체 문서가 있을 때의 정답 예측 점수 (원본 점수)
- $f(q, D \setminus \{s_i\})$: 문장 s_i 하나만 뺐을 때의 정답 예측 점수
- **Δ_i가 크다 = 정답 도출에 결정적인 핵심 문장이다!**

Problem Formulation & RAG Pipeline

Problem Formulation

- RAG 환경에서 언어 모델은 질문 q와 검색된 문서 집합 D를 바탕으로 답변을 생성. $M(y \mid q, D_k)$
- 문제: 검색된 문서 D가 클 경우, Token budget과 연산 비용이 크게 증가.
- 목표: 관련 증거는 보존하고 중복은 제거하는 압축 함수를 통해 짧고 핵심적인 압축 문맥 C를 생성하는 것

$$\arg\max_{\pi} M\left(y \mid q, C_\pi\right), \quad C_\pi = \pi(q, D_k),$$

RAG Pipeline with Compression [Retriever => Compressor => Language Model]

- 증거 보존 (Evidence Retention): 정확한 답변을 위한 필수 정보 유지
- 처리 효율성 (Processing Efficiency): 시스템 부하와 지연(Latency)을 막기 위한 가볍고 빠른 처리
- 토큰 효율성 (Token Efficiency): 토큰 수 감소를 통한 비용 절감 및 추론 속도 향상

Sentence-level Pruning & Leave-One-Out Delta

1. Sentence-level Pruning (문장 단위 가지치기)

- 토큰 단위 압축 시 발생하는 핵심 구문 Fragmentation을 방지하고, 구문과 의미론적 관계를 온전히 보존.
- 검색된 문서를 문장 단위로 분할.

2. Leave-One-Out Delta-based Sentence Classification

- 문장의 중요도를 단독으로 평가하는 대신, "해당 문장을 제거했을 때 전체 문서의 답변 가능성이 얼마나 손실되는가?"를 측정.
- 전체 문맥을 기준으로 변화량(Delta)을 계산하므로, 전반적인 의미 의존성(Semantic dependencies)을 파악할 수 있음.

Formal Loss Function

1. 단서가 포함된 문서 (Clue-filled passages)

- 문서 안에 정답 역할을 하는 핵심 문장이 최소 1개 이상 있을 때 사용합니다.
- 랭킹 손실과 분류 손실을 결합하여 학습.

$$L_{yes} = L_{rank} + \lambda \cdot BCE(p_0, 1)$$

$BCE(p_0, 1)$: 전체 문서를 다 읽었을 때는 정답이 확실히 있다고 가정하고 높은 점수를 예측.

L_{rank} : 각 문장을 뺐을 때 점수가 얼마나 떨어지는지를 보고, 문장들의 중요도 순위를 정확히 매김.

Formal Loss Function

L_{rank}의 3가지 구성 요소:

규칙 A: “핵심 문장 뺀 타격 > 비핵심 문장 뺀 타격” [핵심 문장과 비핵심 문장 간의 점수 격차(Margin)를 크게 벌림]

$$L_{ord} = \sum_{i:y_i=1} \sum_{j:y_j=0} \max(0, m_1 - (\Delta_i - \Delta_j))$$

규칙 B: “핵심 문장 빼면 점수 깎아라!” [핵심 문장 제거 시 점수가 큰 폭으로 하락하도록 유도]

$$L_{crit} = \sum_{k:y_k=1} \max(0, m_2 - \Delta_k)$$

규칙 C: “비핵심 문장 빼면 점수 유지해라!” [비핵심 문장 제거 시 점수 변화가 크지 않도록 페널티 부여]

$$L_{non} = \sum_{k:y_k=0} \max(0, \Delta_k + m_3)$$

Inference Strategy

1. 병렬 처리 및 빠른 필터링

- n개의 문장으로 이루어진 문서는 n+1번의 Forward pass를 독립적이고 병렬적으로 계산하여 속도를 높임.
- 전체 문서의 예측 점수가 임계값 미만이면, 단서가 없다고 판단하여 문서 전체를 즉시 폐기.

2. Adaptive Gap-based Selection

- 단순히 고정된 기준점을 사용하지 않고, 문서마다 유동적으로 핵심 문장을 분류.
- 유의미한 하락폭을 보인 점수들을 내림차순 정렬.
- 점수들 간의 가장 큰 격차(Gap)를 계산.
- 이 최대 격차 지점을 적응형 임계값으로 설정.
- 임계값보다 큰 문장만 최종 핵심 문장으로 선택하여 중복은 거르고 필수 정보만 높은 정밀도로 추출.

Efficiency & Adaptability

Lightweight Encoder-only Model

- LOO-Delta 점수를 계산하려면 문장 개수만큼 모델 추론을 반복해야함.
- LLM 대신 ModernBERT 기반의 가벼운 인코더 전용(Encoder-only) 구조를 채택.
- 메모리 사용량을 최소화하면서, Parallelized Scoring을 통한 Latency감소.

Adaptive Gap-based Selection

- 기존 기술들처럼 "상위 30%만" 혹은 "0.5점 이상만" 남기는 고정된 임계값(Threshold) 사용 X
- 문장들을 Delta 점수 순으로 나열한 뒤, 가장 점수가 뚝 떨어지는 간격(Gap)을 찾아 그 지점을 동적 임계값으로 설정.
- 질문의 난이도나 문서의 정보 밀도에 맞춰, 최적화된 개수의 문장만 유동적으로 추출하여 불필요한 컨텍스트 토큰을 최소화.

KOREA UNIVERSITY

Training Details

1. 학습 데이터 및 라벨링 (Data & Labeling Strategy)

- 학습 데이터셋: HotpotQA
- Augmentation: 문서 내 각 문장을 스캔, 실제 정답 텍스트가 포함된 문장은 Label 1으로, 없는 문장은 Label 0으로 자동 라벨링

2. Hardware & Finetuning

Single NVIDIA RTX 4090 GPU, 21h

LoRA (Rank=64) / bfloat16

KOREA UNIVERSITY

Setup

1. 평가 데이터셋

- LooComp의 범용적인 문맥 압축 능력을 검증하기 위해 5가지 QA 벤치마크를 활용.
- Natural Questions (NQ), TriviaQA, HotpotQA, MuSiQue, 2WikiMultihopQA

2. 비교 모델 및 평가 지표 (Baselines & Metrics)

- 비교 베이스라인: 추출형: LLMLingua, EXIT, Provence 등
- 생성형: RECOMP, CompAct, Refiner 등

주요 평가 지표:

Quality: Exact-Match, F1 Score

Efficiency: 문맥 압축률(Context Saved %) 및 초당 질문 처리 속도(QpS, Latency)

Results

KOREA UNIVERSITY

	NQ				TQA				HQA				2Wiki				Musique			
	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate	EM	F1	Time	Rate
Llama-3.1-8B Instruct																				*Top-5 retrieved chunks*
Raw	35.4	47.3	-	100	60.9	69.8	-	100	30.2	40.9	-	100	23.5	30.2	-	100.0	4.3	10.8	-	100
CompAct	31.4	42.0	2.573	12.1	60.0	69.1	2.594	12.0	28.1	37.5	2.517	11.9	22.1	28.8	2.354	10.6	**6.1**	**13.6**	2.781	11.9
EXIT	**33.8**	45.0	0.359	56.1	60.1	68.2	0.340	50.9	28.9	39.4	0.347	48.6	21.3	27.4	0.380	42.6	4.0	10.8	0.349	45.3
RECOMP-abs	33.5	**45.1**	0.765	**7.8**	53.8	63.5	0.578	**7.6**	28.3	39.4	0.527	**7.0**	**24.4**	**29.6**	0.436	**5.5**	4.3	11.8	0.499	**6.6**
RECOMP-ext	33.8	45.1	**0.009**	47.4	57.4	65.7	**0.009**	47.7	26.7	36.5	**0.010**	50.6	19.2	25.9	**0.008**	50.1	4.2	10.5	**0.009**	48.6
Refiner	33.2	44.9	2.403	15.3	60.3	69.4	1.967	13.2	29.7	40.5	1.624	10.4	22.2	28.2	1.325	8.3	5.3	12.0	1.436	8.9
LongLLMLin	29.8	41.2	0.345	46.2	59.6	68.2	0.344	45.9	28.6	38.5	0.329	47.1	21.5	27.0	0.339	36.9	4.4	10.9	0.347	46.3
Provence	33.4	44.8	0.059	26.4	59.9	69.1	0.056	25.4	28.7	39.4	0.056	19.5	21.5	27.7	0.059	15.5	4.5	11.2	0.056	14.7
Ours	33.4	44.9	0.036	20.0	**60.6**	**69.7**	0.037	20.1	**31.0**	**42.0**	0.038	15.9	22.3	28.8	0.037	12.2	5.7	12.7	0.038	10.8
																				Top-20 retrieved chunks
Raw	37.2	49.3	-	100	64.0	72.7	-	100	31.1	41.7	-	100	25.6	32.0	-	100	4.6	11.1	-	100
CompAct	33.8	45.4	4.312	3.7	60.0	69.0	4.102	3.5	30.9	42.1	4.696	3.7	21.6	28.2	4.905	3.5	5.3	12.4	5.869	4.1
EXIT	**34.6**	45.8	1.414	51.6	58.6	66.4	1.452	45.4	29.9	39.8	1.474	41.9	23.1	28.9	1.619	37.6	4.2	10.9	1.480	41.9
RECOMP-abs	32.9	44.9	1.223	**2.1**	55.9	65.3	1.332	**2.0**	28.5	39.8	1.206	**1.8**	22.4	28.2	1.145	**1.4**	5.2	11.9	1.393	**1.6**
RECOMP-ext	30.9	41.3	**0.036**	11.4	54.8	62.6	**0.037**	11.5	23.3	32.1	**0.035**	12.4	15.8	22.3	**0.036**	12.2	4.0	9.7	**0.037**	12.0
Refiner	31.0	42.0	6.727	8.8	59.1	68.0	5.013	6.5	29.1	39.3	3.915	4.4	22.2	28.1	3.219	3.7	5.1	11.6	4.551	5.0
LongLLMLin	33.4	45.5	1.335	39.0	62.8	70.8	1.436	39.1	30.9	41.7	1.357	39.1	24.9	30.7	1.357	39.3	4.7	11.4	1.343	39.4
Provence	34.2	46.2	0.188	20.8	63.2	71.7	0.189	18.9	30.0	40.5	0.199	13.8	22.5	28.6	0.197	11.9	5.3	11.9	0.196	12.2
Ours	34.4	**46.5**	0.153	13.8	**63.9**	**72.2**	0.161	12.9	**32.4**	**43.7**	0.158	8.5	**25.0**	**30.8**	0.159	6.4	**6.5**	**13.8**	0.164	7.0
Llama-3.3-70B Instruct																				*Top-5 retrieved chunks*
Raw	36.8	51.1	-	100	65.8	75.2	-	100	35.6	47.8	-	100	27.2	34.9	-	100	8.2	16.8	-	100
CompAct	35.3	48.4	2.573	12.1	65.2	74.6	2.594	12.0	36.6	48.6	2.517	11.9	27.0	34.6	2.354	10.6	9.2	17.8	2.781	11.9
EXIT	35.0	48.5	0.359	56.1	64.4	73.8	0.340	50.9	34.2	45.6	0.347	48.6	24.3	31.2	0.380	42.6	7.5	16.1	0.349	45.3
RECOMP-abs	33.9	46.6	0.765	**7.8**	56.9	66.9	0.578	**7.6**	32.2	44.3	0.527	**7.0**	26.7	32.3	0.436	**5.5**	6.5	14.2	0.499	**6.6**
RECOMP-ext	**36.4**	49.1	**0.009**	47.4	65.0	74.1	**0.009**	47.7	32.7	44.1	**0.010**	50.6	22.0	30.0	**0.008**	50.1	7.0	15.3	**0.009**	48.6
Refiner	35.1	48.6	2.403	15.3	65.6	75.2	1.967	13.2	35.8	47.6	1.624	10.4	25.7	32.7	1.325	8.3	8.3	16.6	1.436	8.9
LongLLMLin	32.8	46.3	0.345	46.2	65.7	74.9	0.344	45.9	34.4	46.2	0.329	47.1	25.1	31.8	0.339	36.9	7.9	16.5	0.347	46.3
Provence	35.8	**49.8**	0.059	26.4	66.1	75.6	0.056	25.4	35.2	47.3	0.056	19.5	27.0	33.7	0.059	15.5	6.9	16.3	0.056	14.7
Ours	35.1	48.8	0.036	20.0	**66.8**	**76.3**	0.037	20.1	**37.3**	**50.1**	0.038	15.9	**31.2**	**37.6**	0.037	12.2	**9.5**	**18.8**	0.038	10.8
																				Top-20 retrieved chunks
Raw	39.1	53.4	-	100	68.8	77.9	-	100	38.2	50.3	-	100	30.6	38.5	-	100	10.0	18.9	-	100
CompAct	35.2	48.5	4.312	3.7	64.9	74.0	4.102	3.5	36.1	47.8	4.696	3.7	26.5	33.2	4.905	3.5	8.4	15.9	5.869	4.1
EXIT	**37.0**	51.3	1.414	51.6	66.3	75.8	1.452	45.4	36.2	48.2	1.474	41.9	27.4	34.7	1.619	37.6	8.7	17.7	1.480	41.9
RECOMP-abs	34.2	47.2	1.223	**2.1**	60.2	69.9	1.332	**2.0**	33.1	45.4	1.206	**1.8**	27.7	33.4	1.145	**1.4**	6.7	14.8	1.393	**1.6**
RECOMP-ext	33.5	45.7	**0.036**	11.4	64.0	72.6	**0.037**	11.5	28.3	38.9	**0.035**	12.4	17.1	24.6	**0.036**	12.2	6.0	14.1	**0.037**	12.0
Refiner	33.4	46.4	6.727	8.8	64.1	73.6	5.013	6.5	33.4	45.0	3.915	4.4	25.7	32.0	3.219	3.7	7.8	16.1	4.551	5.0
LongLLMLin	36.2	50.8	1.335	39.0	**67.9**	**77.2**	1.436	39.1	36.6	48.9	1.357	39.1	29.4	37.1	1.357	39.3	8.9	18.2	1.343	39.4
Provence	37.0	**51.3**	0.188	20.8	66.6	76.0	0.189	18.9	35.4	47.5	0.199	13.8	26.3	33.0	0.197	11.9	8.7	18.0	0.196	12.2
Ours	36.3	50.5	0.153	13.8	67.3	76.9	0.161	12.9	**38.5**	**51.2**	0.158	8.5	**31.4**	**37.5**	0.159	6.4	**10.7**	**20.1**	0.164	7.0

	EM	F1	Time	Rate
CompAct	32.2	41.6	3.670	7.7
EXIT	32.0	41.3	0.921	46.2
RECOMP-abs	30.4	39.7	0.910	**4.3**
RECOMP-ext	29.1	38.0	**0.023**	30.4
Refiner	31.6	40.9	3.218	8.5
LongLLMLin	32.3	41.7	0.853	41.8
Provence	32.4	42.0	0.126	17.9
Ours	**34.0**	**43.6**	0.098	12.8

- 기존 Extractive 압축 모델(LLMLingua, EXIT)들을 압도하는 답변 정확도(Exact-Match, F1)를 기록.
- 이는 모델이 단순한 단어 빈도수가 아니라, 정답 도출에 필요한 핵심 문장을 놓치지 않고 완벽하게 골라냈음을 의미.
- 원본 문서의 최대 93% 이상(상위 20개 문서 기준)을 압축하면서도, 초당 처리 지연 시간은 0.2초 미만 달성.
- 속도만 빠르거나 압축만 많이 하는 기존 모델들의 한계를 넘어, 정확도·압축률·속도의 완벽한 타협점을 찾음.

KOREA UNIVERSITY

Robustness at Larger k

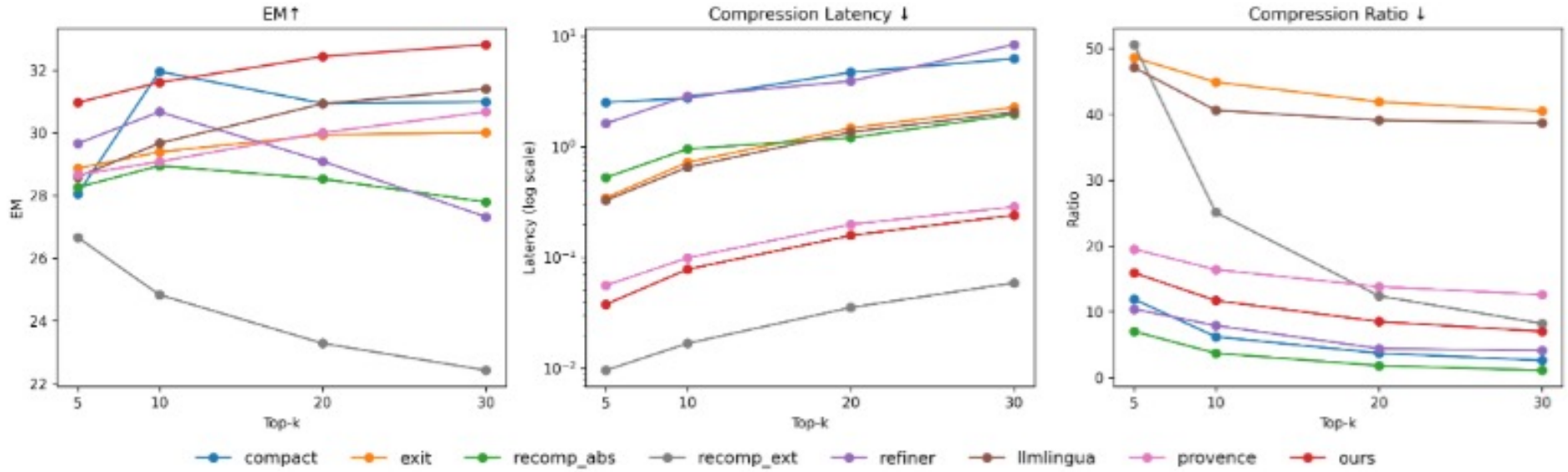


Figure 3: Performance analysis on HQA at increasing top-k = {5, 10, 20, 30} by reader Llama-3.1-8B Instruct, comparing EM, compression latency, and compression ratio between baselines and our proposed method.

- 검색된 문서의 수(k)를 5에서 30까지 늘렸을 때, RECOMP나 Refiner 같은 모델들은 성능이 하락하는 현상을 보이지만, LooComp는 지연 시간은 선형적으로만 증가(0.037s ➔ 0.241s)하면서 EM 점수가 30.97에서 32.82로 꾸준히 상승.
- 입력되는 노이즈가 많아져도 이를 완벽하게 필터링해 내기 때문에, 오히려 정보가 많아져도 Robustness함.

KOREA UNIVERSITY

Analysis

Variants	NQ		TQA		HQA		2Wiki		Musique	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Full	36.8	51.8	69.2	76.7	38.6	51.1	30.8	35.4	10.6	18.8
-BCE	33.7	46.8	67.8	75.9	35.0	46.3	28.1	33.2	8.8	18.3
-crit	35.6	49.3	69.2	76.8	37.6	50.1	29.8	34.0	10.3	17.9
-BCE -crit	32.6	46.4	68.4	76.2	34.6	46.6	24.6	30.1	7.2	17.2

Ablation Study

- 손실 함수의 핵심 요소: 전체 손실 함수에서 BCE 항(문서 전체의 정답 유무 판단)이나 crit을 제거하면 성능이 큰 폭으로 하락.

- 이는 각 손실 함수 요소가 모델의 정확도에 필수적인 시너지를 보여줌

Variants	NQ		TQA		HQA		2Wiki		Musique	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Adaptive-gap	36.8	51.8	69.2	76.7	38.6	51.1	30.8	35.4	10.6	18.8
Margin-based	36.5	50.4	68.7	76.5	38.8	51.2	30.3	34.4	9.9	18.3

Strategy Comparison

1. Margin-based (고정 기준): 학습 때 설정했던 엄격한 커트라인을 추론에도 똑같이 적용하여, 이 점수를 넘지 못하면 무조건 잘라내는 경직된 방식
2. Adaptive-gap : 고정된 점수에 얽매이지 않고, 현재 입력된 문서 내 문장들의 하락폭을 나열한 뒤, 점수 차이(Gap)가 가장 크게 벌어지는 단층 지점을 스스로 찾아내어 유동적으로 핵심을 분리하는 방식.

각 문서의 문맥 흐름에 맞춰 유연하게 판단하는 Adaptive-gap 방식이 모든 데이터셋에서 더 높은 성능을 기록하며 실전에서의 범용성을 입증.

Conclusions

1. 연구 요약 및 의의

- LooComp는 검색 증강 생성(RAG) 시스템의 문맥 압축 효율성을 극대화하기 위해, Leave-One-Out (LOO) 전략과 가벼운 인코더 전용(Encoder-only) 트랜스포머를 결합한 프레임워크.
- 복잡하고 무거운 디코더 기반 LLM 없이도, 문장 단위의 중요도를 정밀하게 계산하고 불필요한 노이즈를 성공적으로 제거할 수 있음을 증명.

2. 한계점 및 향후 과제

- LooComp는 문장 단위로 압축을 수행, 여러 문장에 걸쳐 파편화된 정보를 종합해야만 의미가 파악되는 극도로 복잡한 추론상황에서는 핵심 문장을 일부 놓칠 가능성이 존재.
- 향후 연구에서는 이러한 문장 간의 상호 의존성(Inter-sentence dependencies)을 더 깊이 모델링하는 방향으로 발전이 필요.

3. 실무적 가치

- 단일 도메인(HotpotQA) 데이터만으로 학습했음에도 다양한 데이터셋과 오픈소스/상용 LLM에서 뛰어난 제로샷 일반화(Zero-shot Generalization) 능력을 보여줌.
- 결론적으로 LooComp는 거대 모델 수준의 압축 성능을 제공하면서도 메모리 사용량이 적고 압도적으로 빠른 처리 속도(Latency)를 보장하므로, 실제 상용 RAG 애플리케이션구축에 가장 이상적이고 실용적인 대안임.

Thank You