Long-context LLM

LLM은 정말 Long-context를 Effective & Efficient하게 처리하는가?

Paper 1 Lost in the Middle: How Language Models Use Long Contexts

Paper 2 Efficient Streaming Language Models with Attention Sinks

2026 MOAI Lab AI Study | 20210790 장세림, 20220284 정민호

INTRODUCTION

발표 구성 및 진행 순서

01

Introduction

LLM의 Long-context
처리와 그 한계

02

Paper 1
Lost in the Middle

Lost in the Middle:
How Language Models
Use Long Contexts

03

Paper 2
Streaming LLM

Efficient Streaming
Language Models with
Attention Sinks

04

Conclusion

두 논문의 연관성과
시사점

Long-context LLM의 역설

LLM Context Window의 성장

4K → **1M** → **128K** → **200K+**

GPT-3.5 Turbo (2022) → Gemini 1.5 Pro (2024) → GPT-4o (2024) → Claude 3.7 (2025)

Context Window란?

: 모델이 한 번의 추론에서 동시에 참고할 수 있는 토큰(token)의 최대 길이

→ 모델이 현재 시점에서 "볼 수 있는 텍스트의 최대 범위"

1 지능의 역설

Intelligence Paradox | Lost in the Middle

입력이 길어질수록 모델이 **중간에 위치한 정보를 놓칩니다**.

앞부분(Primacy Bias)과 뒤부분(Recency Bias)에만 집중하여,
실제 장문서 QA 등의 Task에서 성능이 급격히 저하됩니다.

2 비용의 역설

Cost Paradox | KV Cache Explosion

문맥 길이 T에 대해 KV 캐시 메모리와 연산 비용이 **$O(T^2)$**으로 폭증합니다.

실시간 스트리밍과 지속적 대화를 사실상 불가능하게 만드는
핵심 장벽입니다.

"LLM이 Long-context를 입력받고 처리할 수 있다고 해서, 그것을 효과적으로 활용하는 것은 아니다"

INTRODUCTION

LLM은 Long-context를 실제로 잘 활용하는가?

"LLM은 Long-context input을 효과적이고 효율적으로 활용하는가?"

Paper 1

Lost in the Middle

Lost in the Middle: How Language Models Use Long Contexts

핵심 발견

현재 LLM은 긴 문맥 내 중간 정보를 robust하게 활용하지 못함.

Primary & Recency Bias

문맥의 맨 앞(Primacy)과 맨 뒤(Recency) 정보는 잘 활용하지만,
중간에 위치한 정보는 무시되는 경향이 뚜렷하게 나타남.

Paper 2

Efficient Streaming Language Models

Efficient Streaming Language Models with Attention Sinks

핵심 발견

Attention Sink를 활용해 streaming 배포를 안정적으로 가능하게 함.

Attention Sink

초기 토큰(Initial Tokens)에 의미와 무관하게 과도한 Attention 점수가 집중되는 현상이 발생함.

이를 활용해 streaming inference를 안정화함.

두 논문은 모두 "초기 토큰 편향"이라는 공통된 현상을 다른 관점에서 조명함 — 하나는 한계로, 하나는 활용 방안으로

01

PAPER 1

Lost in the Middle

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

논문 개요

LLM이 더 많은 context를 입력받는다고, 그만큼 성능이 향상되는가?

01 핵심 문제

Context Window 확장 ≠ 긴 문맥 활용 능력 향상

02 핵심 현상

앞·뒤 정보는 잘 쓰지만, 중간 정보 활용 성능이 급락

03 실험 설계

Multi-Document QA와 Key-Value Retrieval로 위치 효과를 통제

04 논문 의의

Long-context 모델은 위치-불변성(Position-Invariance)으로 평가해야 함

📌 **이 논문은 Long-context LLM의 “효과성” 문제를 실증적으로 보여준다.**

정답 문서의 위치가 바뀌면 성능도 바뀌는가?

Input Context

```
Write a high-quality answer for the given question using only the provided search
results (some of which might be irrelevant).

Document [1](Title: Asian Americans in science and technology) Prize in physics for
discovery of the subatomic particle J/ψ. Subrahmanyan Chandrasekhar shared...
Document [2](Title: List of Nobel laureates in Physics) The first Nobel Prize in
Physics was awarded in 1901 to Wilhelm Conrad Röntgen, of Germany, who received...
Document [3](Title: Scientist) and pursued through a unique method, was essentially
in place. Ramón y Cajal won the Nobel Prize in 1906 for his remarkable...

Question: who got the first nobel prize in physics
Answer:
```

Desired Answer

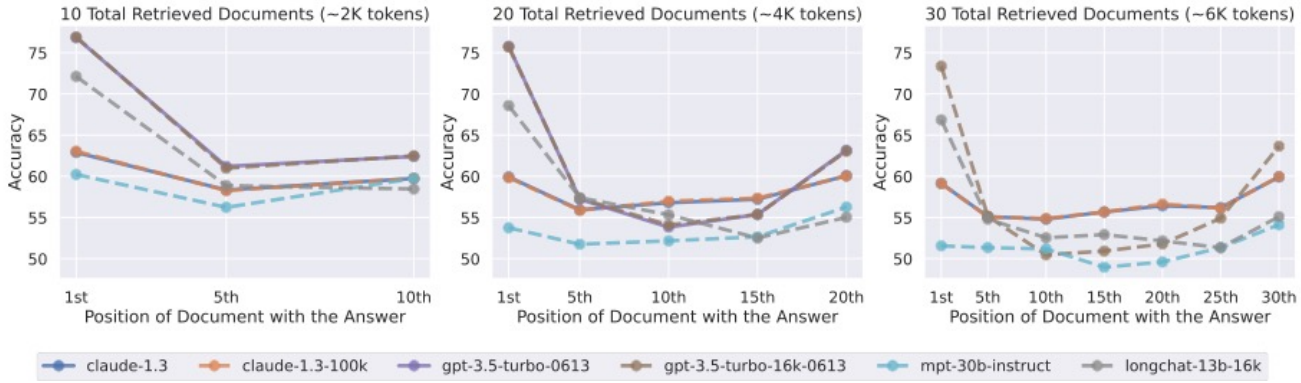
```
Wilhelm Conrad Röntgen
```

실험 구조

정답 문서 1개 + distractor 문서 k-1개를 입력 context에 배치

위치 조작

정답 문서를 1st / middle / last 위치로 이동시키며 성능 측정



Model	Closed-Book	Oracle
LongChat-13B (16K)	35.0%	83.4%
MPT-30B-Instruct	31.5%	81.9%
GPT-3.5-Turbo	56.1%	88.3%
GPT-3.5-Turbo (16K)	56.0%	88.6%
Claude-1.3	48.3%	76.1%
Claude-1.3 (100K)	48.2%	76.4%

📌 **긴 context 안에서도 LLM은 정보를 균등하게 활용하지 못한다.**

단순 검색 과제에서도 “중간 정보”를 놓치는가?

Input Context

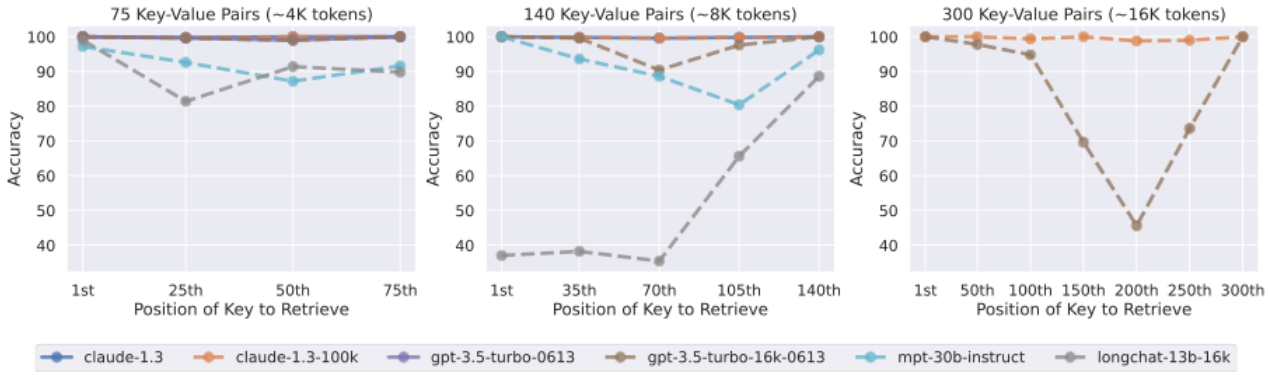
```
Extract the value corresponding to the specified key in the JSON object below.

JSON data:
{"2a8d601d-1d69-4e64-9f90-8ad825a74195": "bb3ba2a5-7de8-434b-a86e-a88bb9fa7289",
 "a54e2eed-e625-4570-9f74-3624e77d6684": "d1ff29be-4e2a-4208-a182-0cea716be3d4",
 "9f4a92b9-5f69-4725-ba1e-403f08dea695": "703a7ce5-f17f-4e6d-b895-5836ba5ec71c",
 "52a9c80c-da51-4fc9-bf70-4a4901bc2ac3": "b2f8ea3d-4b1b-49e0-a141-b9823991ebeb",
 "f4eb1c53-af0a-4dc4-a3a5-c2d50851a178": "d733b0d2-6af3-44e1-8592-e5637fdb76fb"}

Key: "9f4a92b9-5f69-4725-ba1e-403f08dea695"
Corresponding value:
```

Desired Output

```
703a7ce5-f17f-4e6d-b895-5836ba5ec71c
```



왜 이 실험이 중요한가?

자연어 의미·추론을 제거하고, “입력에서 정확히 찾기” 능력만 검사

관찰 결과

Claude는 거의 완벽하지만 GPT-3.5, MPT 등은 중간 위치에서 성능 하락

📌 **의미 추론이 거의 없는 과제에서도 위치 편향이 나타난다.**

왜 LLM은 중간 정보를 robust하게 쓰지 못하는가?

1. Model Architecture

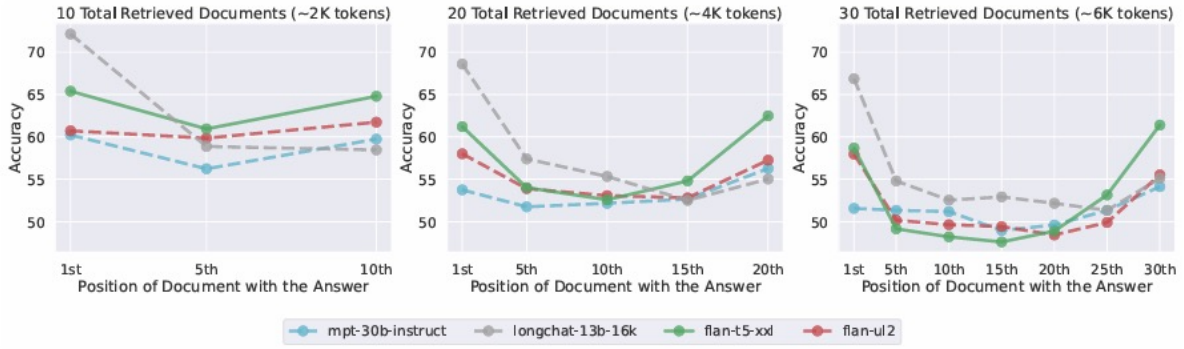
Encoder-decoder는 훈련 길이 안에서는 안정적이지만, 훈련 길이를 넘어서면 다시 U-shaped curve 발생

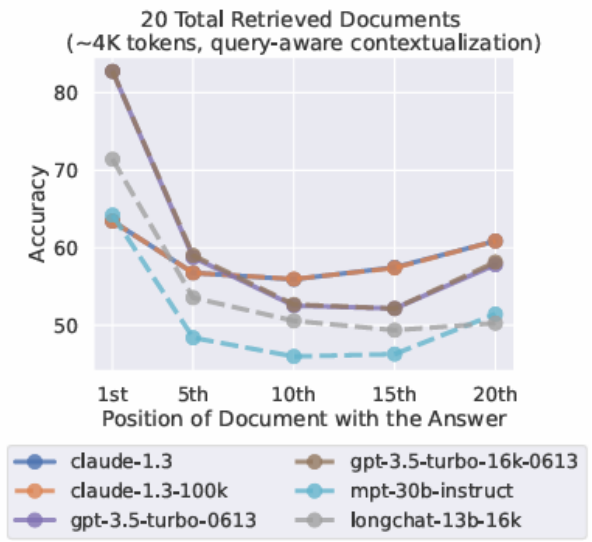
2. Query-Aware Contextualization

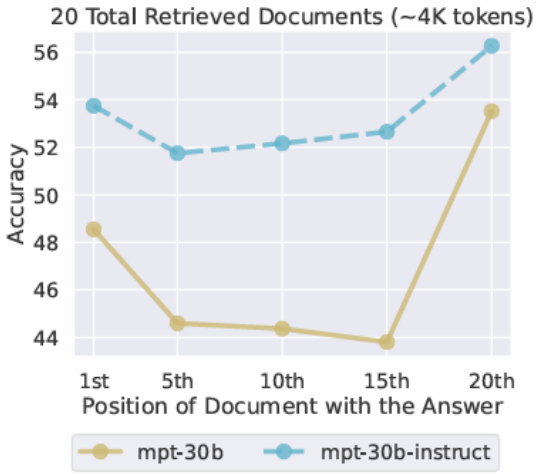
질문을 앞·뒤에 배치하면 Key-Value retrieval은 개선되지만, Multi-doc QA는 거의 개선되지 않음

3. Instruction Fine-Tuning

Base model도 U-shaped curve를 보임 → instruction tuning만의 문제가 아님

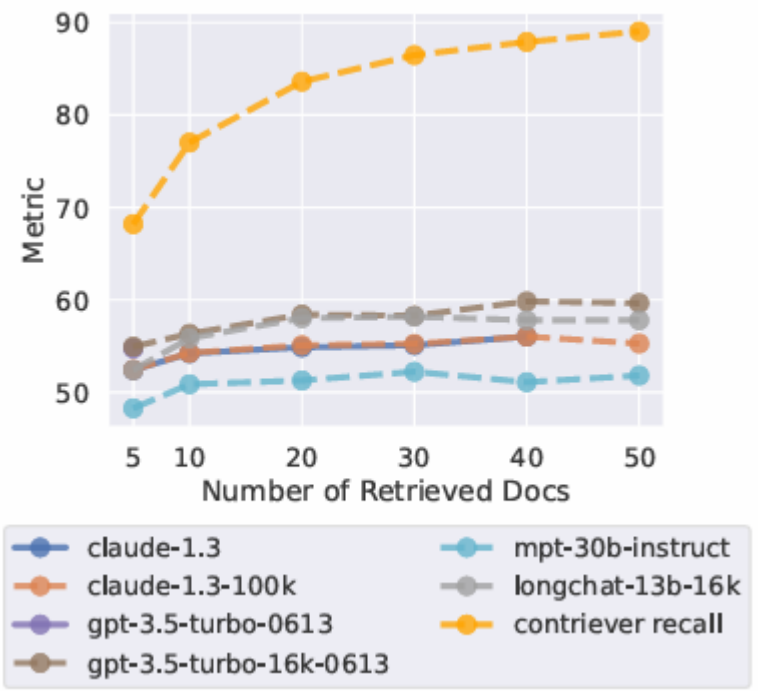






📌 **특정 프롬프트나 fine-tuning만으로 설명되지 않는 구조적 위치 편향 문제다.**

Is More Context Always Better?



실험 구조

Retriever가 Wikipedia에서 k개 문서를 가져오고, LLM reader가 답변 생성

핵심 관찰

문서 수가 늘면 retriever recall은 증가하지만, reader accuracy는 빠르게 포화

성능-비용 trade-off

20개 이후 추가 문서는 성능 개선이 작지만 latency와 cost는 증가

실전적 시사점

무작정 더 많이 넣기보다 reranking / truncation이 필요

More Context ≠ **Better Performance**

📌 **더 많은 context는 정보량을 늘리지만, 동시에 noise와 reasoning 부담도 늘린다.**

Context Window Size 확장이 활용 능력 향상을 보장하지 않는다

01 Position-Invariance 평가

정답 정보의 위치를 앞·중간·뒤로 바꾸어도 성능이 안정적인지 확인해야 함

02 Retrieval-Augmented Generation

검색 문서를 많이 넣는 것보다 relevant information을 어디에 배치할지가 중요

03 Long-context 연구 방향

긴 입력을 받는 능력과 긴 입력을 활용하는 능력을 분리해서 평가해야 함

Paper 1

Lost in the Middle

Long-context input을
효과적으로 활용하지 못함

Paper 2

StreamingLLM

긴 입력을 효율적으로
streaming 처리하는 프레임워크

공통 문제의식

LLM은 Long-context input을 효과적이고 효율적으로 활용하는가?

→ Lost in the Middle은 "활용 실패"를 진단하고, StreamingLLM은 "효율적 처리"의 우회 전략을 제시한다.

📌 **이제 두 번째 논문은 Attention Sink를 활용해 streaming inference를 안정화하는 방법을 제안한다.**

02

PAPER 2

Streaming LLM

Efficient Streaming Language Models with Attention Sinks (Xiao et al., 2024)

논문 개요

이 논문은 초기 토큰(Attention Sink)에 집중되는 Attention 패턴을 발견하고,
이를 활용한 StreamingLLM 프레임워크를 제안합니다.

1 **Attention Sink 현상**

초기 토큰에 의미와 무관하게 과도한 Attention이 집중되는 현상 발견.

2 **Window Attention의 한계**

기존 sliding window 방식은 초기 토큰 제거 시 perplexity가 폭발적으로 증가.

3 **StreamingLLM 프레임워크**

Attention Sink 토큰 + 최근 토큰 KV Cache 유지로 무한 길이 streaming 가능.

4 **재학습 없는 배포**

기존 모델 그대로 streaming 환경에 적용 가능 — 추가 fine-tuning 불필요.

Longer Sequence 처리의 중요성과 문제점

Longer Sequence 처리의 중요성

- LLM은 대화 시스템, 문서 요약, 코드 완성, 질의응답 등 다양한 NLP 응용에 활용됨.
- 이상적인 LLM은 장문 대화나 긴 컨텍스트에서도 정확한 시퀀스 생성을 수행해야 함.

Longer Sequence 처리의 문제와 해결 노력

LLM은 pretraining에 활용된 attention window 크기에 제한된다

1. Attention window size 확장
2. Training 개선
3. Inference Efficiency 개선

→ 처리 가능한 시퀀스 길이는 여전히 본질적으로 유한함

핵심 질문

"Can we deploy an LLM for infinite-length inputs without sacrificing efficiency and performance?"

Infinite Input Stream의 두 가지 도전

① KV Cache 메모리 폭증
: decoding 단계에서 모든 이전 토큰의 Key-Value state를 캐싱
→ 메모리 급증 및 decoding 지연

② Extrapolation 실패
: 사전학습 시퀀스 길이를 초과하면 성능이 급격히 저하됨

📌 Streaming 환경에서 LLM의 persistent deployment를 위해서는 무한 시퀀스를 효율적으로 처리할 수 있는 새로운 framework가 필요함

LLM의 Long Text Utilization을 위한 연구 방향

1 Length Extrapolation

정의 훈련 시 짧은 텍스트로 학습된 LLM을 추론 시 더 긴 시퀀스에서도 동작하게 하는 연구 방향

RoPE | ALiBi

→ **한계**

Pretraining 대비 input context 크게 증가하면 성능 저하
→ no existing LLMs to fit for streaming applications

2 Context Window Extension

정의 LLM의 컨텍스트 윈도우 자체를 늘려 단일 forward pass에서 더 많은 토큰을 처리하는 연구 방향

FlashAttention | Approximate attention methods

→ **한계**

모든 기법이 context window를 제한적으로만 확장
→ limitless inputs 처리 불가

StreamingLLM과 보완적 통합 가능

3 Improving LLMs' Utilization of Long Text

정의 긴 텍스트 입력받는 것을 넘어 컨텍스트 내 내용을 잘 파악·활용하도록 LLM을 최적화하는 연구 방향

Long-context understanding 향상 =/= Streaming deployment 효율화

→ **한계**

Length Extrapolation과 Context Window Extension이
반드시 긴 context를 능숙하게 활용한다는 의미가 아님

Streaming LLM

⇒ 긴 스트리밍 입력/출력을 제한된 메모리로 계속 처리하게 하는 inference optimization 기술

: 추론 능력이 향상된 것이 아니라 장문맥추론이 가능한 효율적인 방법을 개발한 것

본 연구의 3가지 핵심 내용

01

Attention Sink 발견

LLM은 언어 모델링과 무관하게 초기 토큰에 비정상적으로 높은 Attention Score를 집중시킨다.
이 현상을 'Attention Sink'로 명명하고, Softmax 연산의 구조적 특성에서 원인을 규명한다.

02

StreamingLLM 프레임워크 제안

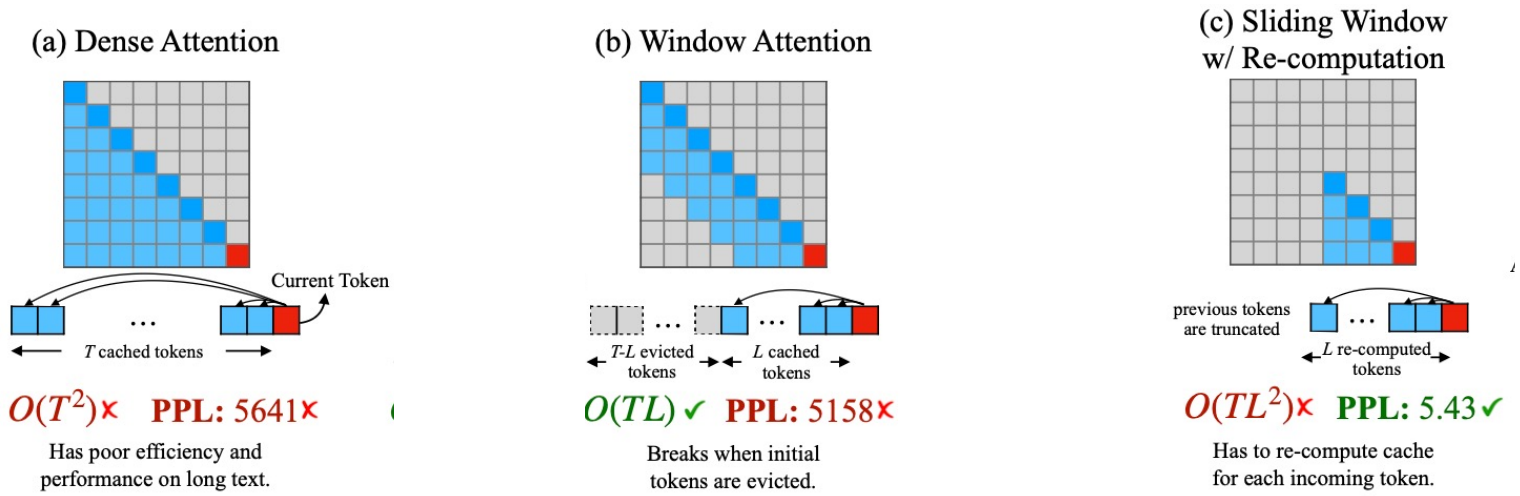
파인튜닝 없이 이미 학습된 LLM이 무한 길이의 입력을 처리할 수 있게 하는 프레임워크.
Attention Sink 토큰의 KV를 Rolling KV Cache와 결합하여 최대 4백만 토큰 처리를 구현한다.

03

전용 Sink Token Pretraining 제안

모든 학습 샘플 앞에 학습 가능한 전용 Sink Token을 추가하는 Pretraining 전략.
단 1개의 Sink Token만으로 안정적인 Streaming 성능을 달성하고, 일반 NLP 성능도 동등하게 유지한다.

Streaming LLM vs 기존 Attention Methods



Dense Attention

모든 이전 토큰 KV 캐싱
→ 최고 품질

메모리 Quadratic 증가,
긴 시퀀스에서 성능 저하

Window Attention

최근 L개 토큰 KV만 유지
→ 메모리 고정

초기 토큰 제거 시 모델 붕괴
(PPL 5158배 급등)

Sliding Window w/ Re-computation

각 토큰 생성 시마다 최근 L개 토큰에 대한 KV 재게산 → 품질 유지

연산 비용 Quadratic 증가
→ 매우 느림

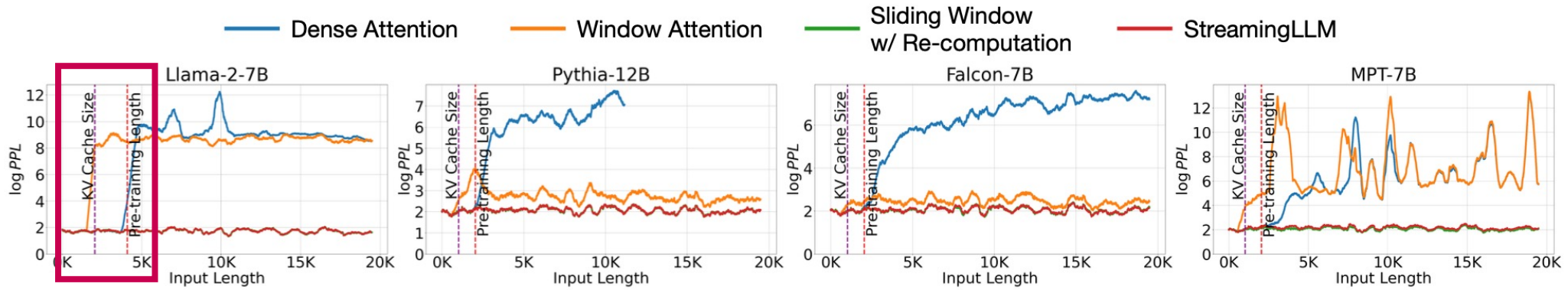
Streaming LLM

Attention sink(초기 토큰) KV
+ 최근 윈도우 KV 유지

22.2× 속도 향상 (Sliding Window w/ Recomputation 대비)

📌 기존 Attention Methods는 Long-context 처리 시 품질(PPL)과 효율(연산 속도·메모리)을 모두 만족하지 못함

기존 Attention Methods 실패 분석



Dense Attention

모든 이전 토큰 KV 캐싱
→ 최고 품질

pre-training length 넘으면
성능 붕괴

Window Attention

최근 L개 토큰 KV만 유지
→ 메모리 고정

cache size 넘으면
(=초기 토큰 제거되면) 성능 붕괴

Sliding Window w/ Re-computation

각 토큰 생성 시마다 최근 L개 토큰에 대한 KV 재게산 → 품질 유지

PPL은 안정적이지만 복잡도가 큼

Streaming LLM

Attention sink(초기 토큰) KV
+ 최근 윈도우 KV 유지

sliding window with recomputation baseline과 거의 동일

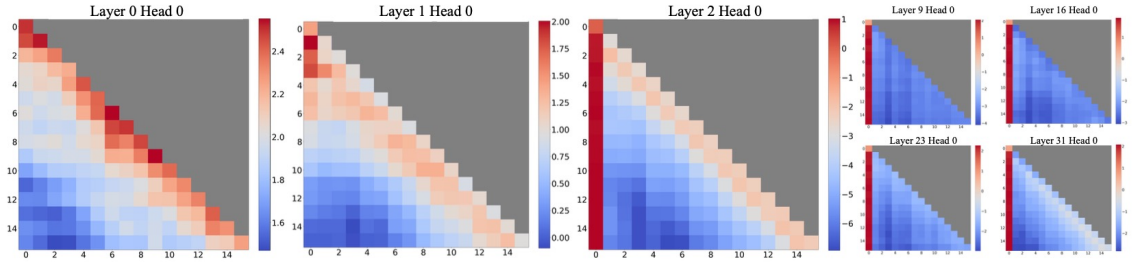
📌 Window Attention은 초기 토큰이 Evict되는 순간 PPL이 붕괴한다 → 초기 토큰의 KV가 LLM 안정성의 Anchor일 수 있다

Attention Sink: Initial Tokens 제거 시 LLM 붕괴 이유

Attention Sink 정의

의미론적 중요도와 무관하게 초기 토큰이 각 layer head에서 과도한 Attention Score를 흡수하는 현상.

→ 초기 토큰이 다른 토큰들의 '불필요한 Attention 쓰레기통(dump)' 역할을 하게 됨.



[Fig2. Visualization of the average attention logits in Llama-2-7B]

1 왜 Attention Sink가 발생할까?

Softmax는 모든 토큰에 대한 Attention Score 합이 반드시 1이 되어야 함.

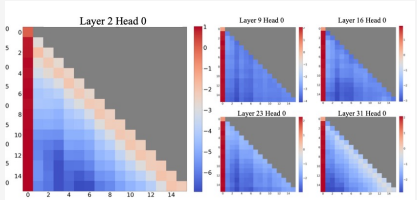
현재 쿼리가 이전 토큰과 강한 의미적 매칭이 없어도, 모델은 불필요한 Attention 값을 어딘가에 반드시 할당해야 함.

$$\text{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^{N} e^{x_j}}, \quad x_1 \gg x_j, j \in 2, \ldots, N$$

2 왜 초기 토큰에 sink 될까?

초기 토큰은 Autoregressive 특성상 이후 모든 토큰에서 참조 가능함. 이 전역 가시성 덕분에 가장 쉽게 Sink 역할을 수행하도록 학습될 것

사전학습 시 모든 샘플에 일관된 시작 토큰이 없어 여러 초기 토큰을 Sink로 사용하도록 학습될 것



3 초기 토큰의 다른 요소가 원인은 아닐까?

초기 4개 토큰을 줄바꿈("\n")으로 대체해도 PPL이 동등하게 복원됨

→ 의미론적 내용이 아닌 절대적 위치가 Sink 역할의 본질일 것

Llama-2-13B	PPL (↓)
0 + 1024 (Window)	5158.07
4 + 1020	5.40
4"\n"+1020	5.60

[Table 1. Effects of initial tokens]

Streaming LLM: Rolling KV Cache with Attention Sinks

Streaming LLM의 KV Cache = ① Attention Sink 영역 (초기 4개 토큰) + ② Rolling KV Cache (최근 N개 토큰) → Fine-tuning 불필요

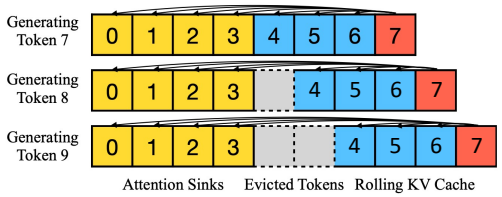


Figure 4: The KV cache of StreamingLLM.

Cache Config	0+2048	1+2047	2+2046	4+2044	8+2040
Falcon-7B	17.90	12.12	12.12	12.12	12.12
MPT-7B	460.29	14.99	15.00	14.99	14.98
Pythia-12B	21.62	11.95	12.09	12.09	12.02
Cache Config	**0+4096**	**1+4095**	**2+4094**	**4+4092**	**8+4088**
Llama-2-7B	3359.95	11.88	10.51	9.59	9.54

[Table 2. Effects of the number of initial tokens]

① Attention Sink 영역 (초기 4개 토큰)

초기 토큰의 KV 상태를 항상 캐시에 보존. Softmax attention 분포를 안정적으로 유지하는 attention sink(Anchor) 역할 수행.

② Rolling KV Cache (최근 N개 토큰)

가장 최근 토큰의 KV 유지하며 새 토큰 생성 시 가장 오래된 토큰을 제거해 언어 모델링에 필수적인 로컬 컨텍스트를 보존.

설계 포인트

Finetuning 불필요

StreamingLLM의 디자인은 범용적

어떠한 model finetuning 없이도 window attention의 perplexity 복원 가능

Positional Embedding 부여 방식

원래 텍스트 상 위치가 아닌 cache 내부 기준으로 positional embedding 부여

Relative positional encoding (RoPE, ALiBi)을 사용하는 모든 Autoregressive LLM에 통합 가능한 디자인

Extrapolation 가능

모델의 pre-training 당시 attention window size를 넘어서는 길이에서도 효율적으로 작동하도록 보장 (지금 보고 있는 캐시 내부만 관리하면 되기 때문에)

Pre-Training LLMs with a Attention Sink

핵심 제안: 모든 학습 샘플 앞에 전용 Learnable Sink Token 추가해 Pretraining 하자 → 단 1개로 안정적 Streaming 달성, 일반 NLP 성능 손실 없음

기준 모델

Vanilla

표준 SoftMax Attention

일반 SoftMax 사용.

불필요한 Attention을 흡수할 전용 Sink가 없어 초기 토큰 여러 개를 자연스럽게 Sink로 활용함.

Cache 0+1024	PPL 27.87
Cache 1+1023	PPL 18.49
Cache 4+1020	**PPL 18.05**

Streaming 안정화에 초기 토큰 4개 필요

부분 개선

$$\text{SoftMax}_1(x)_i = \frac{e^{x_i}}{1 + \sum_{j=1}^{N} e^{x_j}}$$

Zero Sink

SoftMax-off-by-One (SoftMax_1) (Miller, 2023)

분모에 1을 추가한 SoftMax_1 사용.
(K,V가 모두 0인 토큰을 Prepend하는 것과 동일)

Attention Sink 문제를 일부 완화, 완전하지 않음.

Cache 0+1024	PPL 29214
Cache 2+1022	PPL 18.27
Cache 4+1020	**PPL 18.01**

여전히 여러 초기 토큰 필요
(모델 입장에서 '0'은 어떠한 Feature도 없는 상태)

최적 방법

Learnable Sink Token

전용 학습 가능 Placeholder 추가

모든 학습 샘플 앞에 학습 가능한 Sink Token 1개를 명시적으로 삽입.

불필요한 Attention을 이 토큰 하나로 집중시켜, 단 1개의 Sink Token만으로 안정화 완성.

Cache 0+1024	PPL 1235
Cache 1+1023	**PPL 18.01**
Cache 4+1020	**PPL 18.02**

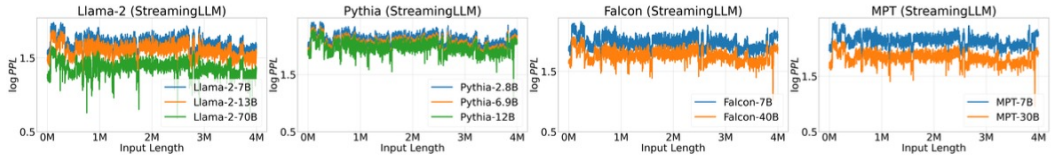
단 1개 Sink Token으로 완전 안정화

StreamingLLM 실험 결과

LANGUAGE MODELING ON LONG TEXTS ACROSS LLM FAMILIES AND SCALES

실험 조건

- **평가 데이터셋:** PG19 테스트셋 (100권 연결, ~4M 토큰)
- **캐시 크기:** Llama-2 → 2048, Falcon/Pythia/MPT → 1024
- **비교 대상:** Dense Attention, Window Attention



[Fig5. Language modeling perplexity of StreamingLLM with 4M+ tokens]

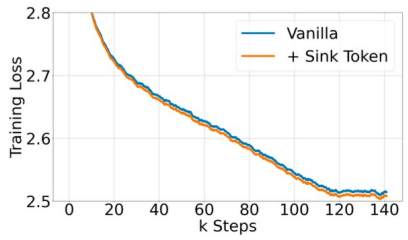
실험 결과 및 결론

- Dense / Window Attention은 일정 길이 초과 시 perplexity 급등
- **StreamingLLM은 4M 토큰 이상**의 초장문에서도 perplexity가 **안정적으로 유지**
- 다양한 규모 및 아키텍처의 LLM 모델에서 일관된 성능 확인
  (Llama-2-[7,13,70]B, Falcon-[7,40]B, Pythia-[2.8,6.9,12]B, MPT-[7,30]B 등)

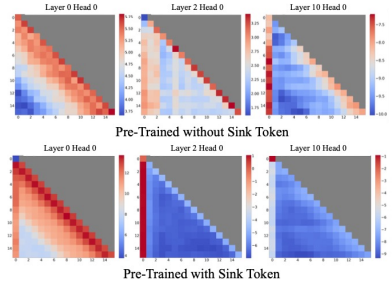
RESULTS OF PRE-TRAINING WITH A SINK TOKEN

실험 조건

- **모델:** 160M 파라미터 LM 두 모델 (Pythia-160M 기반)
- **학습 설정:** 동일 조건, 한 모델만 매 샘플 앞에 sink token 추가
- **학습 스텝:** 143,000 steps / 8×A6000 GPU



[Fig6. Pre-training loss curves of models]



[Fig7. Visualization of average attention logits]

실험 결과 및 결론

- Sink token 유무에 따른 pre-training loss 곡선이 유사한 수렴 추세를 보임
- Sink token이 존재하면 모든 레이어, 헤드에서 sink token에 attention이 집중됨
- **결론:** 학습 단계에서 단일 sink token 도입 → streaming 성능 개선, 기존 성능 유지

StreamingLLM 실험 결과

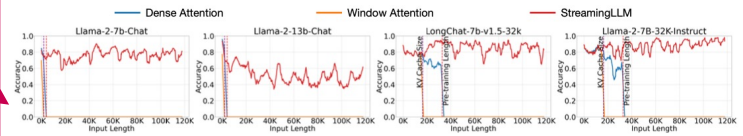
STREAMING QUESTION ANSWERING WITH INSTRUCTION-TUNED MODELS

실험 조건

- **데이터셋:** ARC-[Easy, Challenge] 스트리밍 연결
- **모델:** Llama-2-[7,13,70]B-Chat, 캐시 크기 1024
- **추가:** StreamEval (10줄마다 쿼리, 120K 토큰)

Model	Llama-2-7B-Chat		Llama-2-13B-Chat		Llama-2-70B-Chat	
Dataset	Arc-E	Arc-C	Arc-E	Arc-C	Arc-E	Arc-C
One-shot	71.25	53.16	78.16	63.31	91.29	78.50
Dense			OOM			
Window	3.58	1.39	0.25	0.34	0.12	0.32
StreamingLLM	71.34	55.03	80.89	65.61	91.37	80.20

[Table5. Accuracy (in %) on the ARC-[Easy, Challenge] datasets]



[Fig9. Performance on the StreamEval benchmark]

실험 결과 및 결론

- **Dense Attention**: Pre-trained Window 초과 시 붕괴
- **Window Attention**: Cache Size 초과 시 정확도 급락
- **StreamingLLM**
  - one-shot 기준선과 동등한 정확도 유지
  - StreamEval에서 120K 토큰까지 안정적 정확도 유지

ABLATION STUDIES

실험 조건 1 | Number of Initial Tokens

실험 결과 및 결론

- **1~2개:** perplexity 불완전 회복
- **4개:** 성능 충분히 복원 → 기본값으로 채택
- **8개 이상:** 추가 효과 미미

실험 조건 2 | Cache Sizes

Cache	4+252	4+508	4+1020	4+2044
Falcon-7B	13.61	12.84	**12.34**	12.84
MPT-7B	**14.12**	14.25	14.33	14.99
Pythia-12B	13.17	12.52	**12.08**	12.09

Cache	4+508	4+1020	4+2044	4+4092
Llama-2-7B	9.73	9.32	**9.08**	9.59

[Table6. Effects of cache size on StreamingLLM's performance]

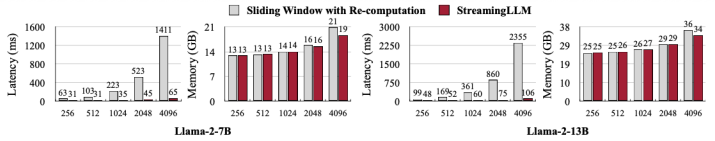
실험 결과 및 결론

- 캐시 크기와 perplexity가 일관되지 않음
  → LLM이 제공된 컨텍스트를 완전히 활용하지 못함을 시사

EFFICIENCY RESULTS

실험 조건

- **비교 대상:** Sliding Window w/ Re-computation
- **측정 지표:** 토큰당 Decoding Latency & 메모리 사용량
- **환경:** NVIDIA A6000 GPU / Llama-2-7B & 13B



[Figure10. Decoding latency and memory usage]

실험 결과 및 결론

Latency

- StreamingLLM: 캐시 크기에 따라 선형 증가
  → 최대 22.2× 속도 향상 (4096 기준)
- Re-computation: 캐시 크기에 따라 이차 증가

Memory

- 두 방법 모두 동일 수준 유지 → StreamingLLM이 속도를 높이면서도 메모리 부담 없음

Streaming LLM 결론

01

핵심 문제

스트리밍 환경에서 LLM 배포는 KV 캐시의 과도한 메모리 소비와 학습 길이를 초과하는 텍스트에서의 성능 저하라는 두 가지 핵심 과제를 안고 있음.

02

Attention Sink 현상 발견

초기 토큰들이 의미론적으로 중요하지 않음에도 불구하고 매우 높은 attention score를 받는 "attention sink" 현상을 발견.
Window Attention 실패의 근본 원인으로 규명.

03

StreamingLLM 프레임워크 제안

Fine-tuning 없이 유한한 attention 윈도우로 학습된 LLM이 무한 길이 텍스트를 처리할 수 있도록 attention sink 토큰과 rolling KV 캐시를 결합한 경량 프레임워크를 도입.

04

LLM의 Streaming Deployment 가능성

StreamingLLM은 LLM의 pre-training context window size와 실제 생성 길이를 분리해 스트리밍 배포의 새로운 가능성을 열었음.

CONCLUSION

Long-context LLM 향후 연구 방향

Lost in the Middle

성능의 U자형 곡선 관찰
→ 현상 실증

공통 원인

SoftMax의 구조적 편향
초기 토큰 과도 집중 (Attention Sink)

StreamingLLM

Attention Score 분포 시각화
→ 메커니즘 발견 + 프레임워크 제안

" Context Window Size 확장이 활용 능력 향상을 보장하지 않는다 "

Context Window Size라는 단일 지표가 실질적 성능을 대표하지 못한다

01

위치-불변성 평가

정보 위치를 다양하게 바꾸는
Position-Invariance 평가를
표준 벤치마크에 포함할 수 있다

02

아키텍처 편향 해결

SoftMax 구조적 편향을 근본적으로
해결하는 새로운 Attention
메커니즘 연구가 필요하다

03

단기 해법 + 장기 혁신

Attention Sink 재활용은
훌륭한 우회책이나, 근본 문제는
여전히 열려 있다

Long-context LLM

LLM은 정말 Long-context를 Effective & Efficient하게 처리하는가?

감사합니다

2026 MOAI Lab AI Study | 20210790 장세림, 20220284 정민호