Efficient Streaming Language Models With Attention Sinks

(ICLR 2024)

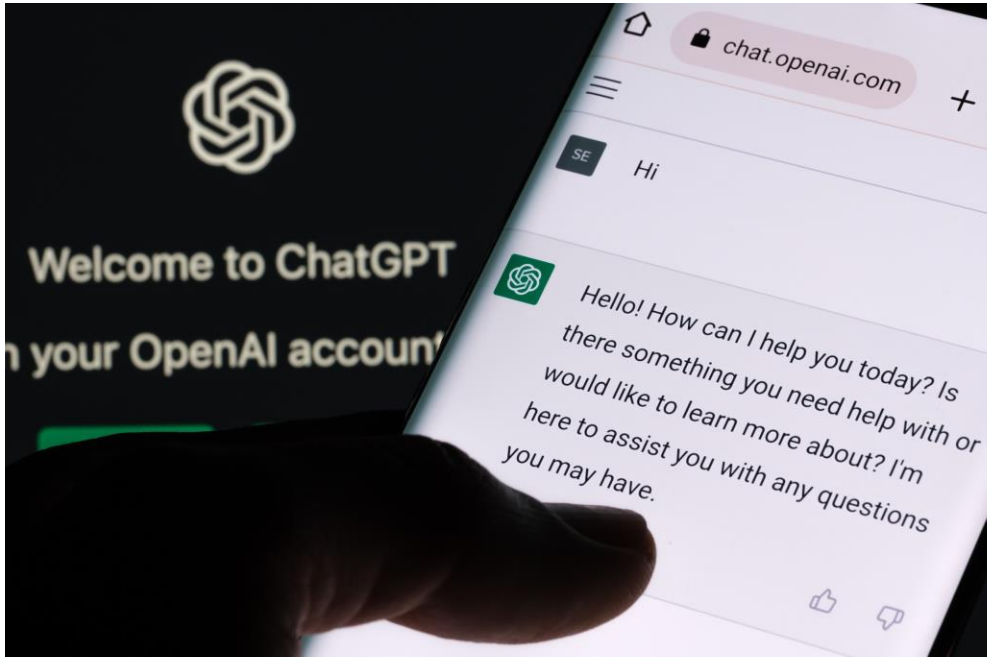
ILLINOIS

Speaker: Ya-Ting Pai

CS598 AI Efficiency

Background

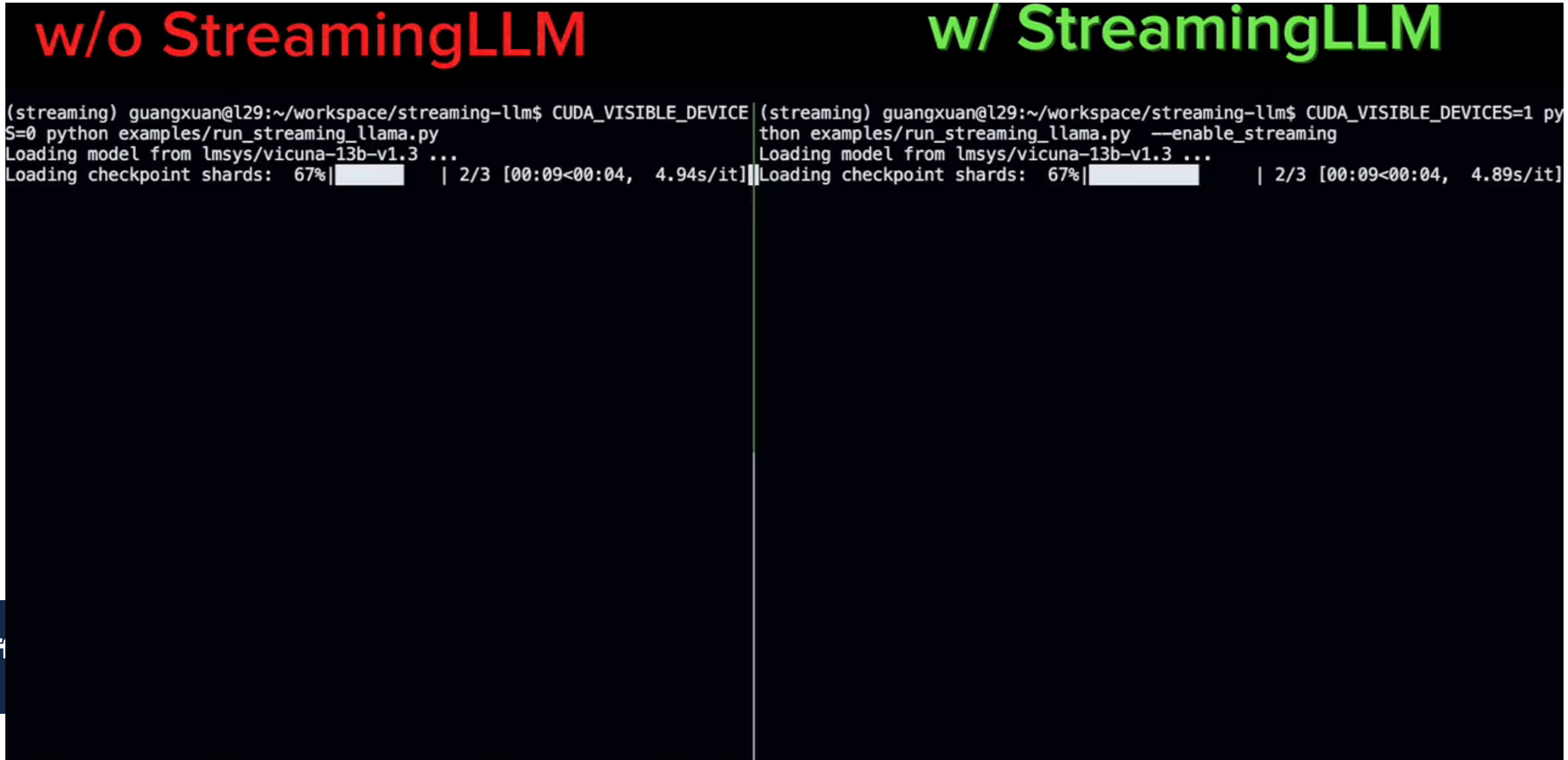
- Streaming devices that need multi-round interactions



LLM for Infinite-length inputs

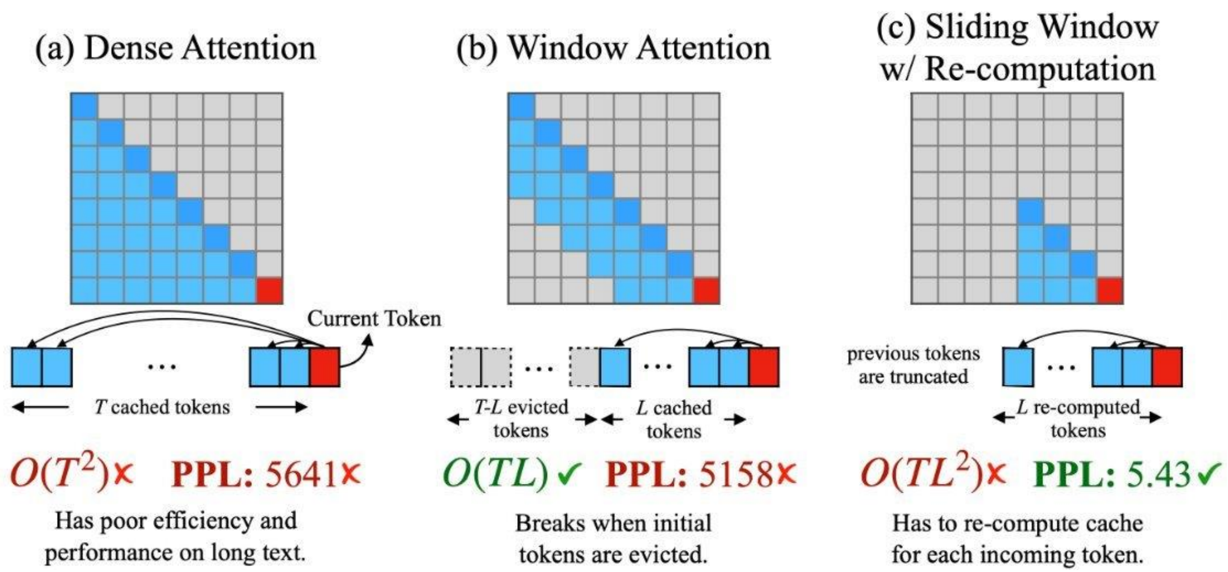
Challenges

- Don't have extensive memory
- Can't generalize to longer texts than the context length



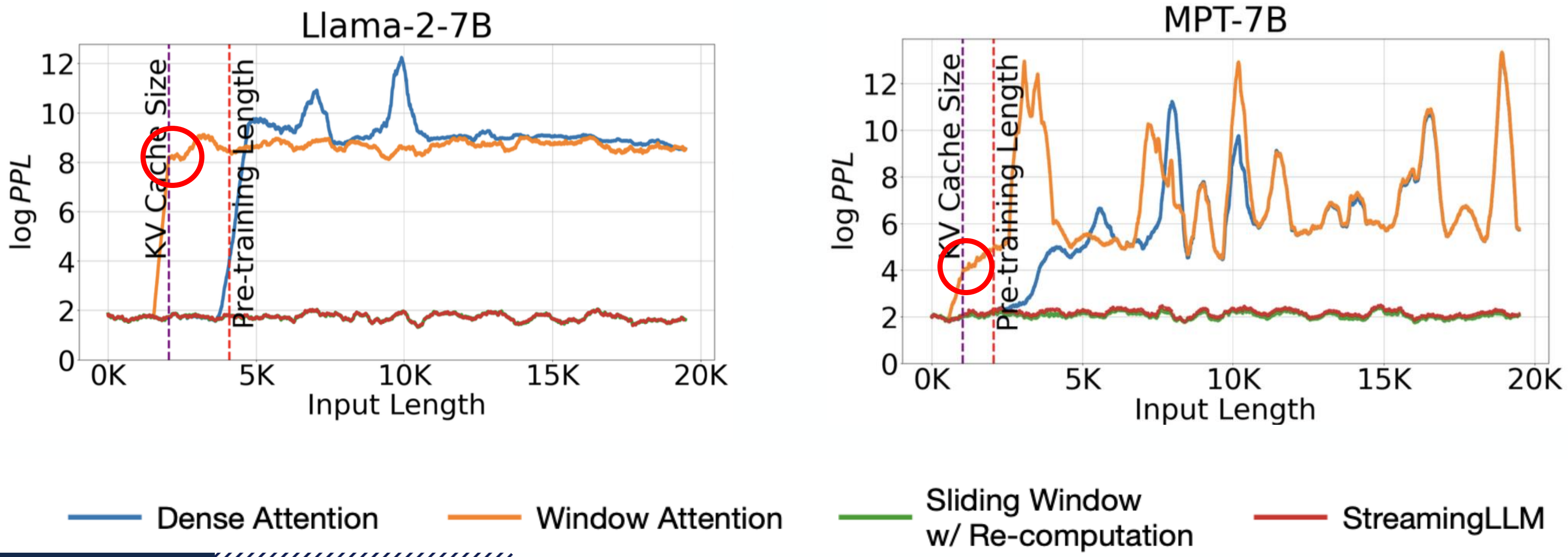
Related Work

- **Dense Attention:** cache the Key and Value states of all previous tokens
- **Window Attention:** caches the most recent L tokens' KV
- **Sliding Window Attention:** rebuilds the KV states from the L recent tokens for each new token



Point of Perplexity Surge

- The lower the perplexity, the better the model is at guessing what is next
- Perplexity spikes when the text length surpasses the cache size



Attention Sinks

- Initial tokens receive high attention scores.
- Softmax normalizes attention scores to sum to 1.

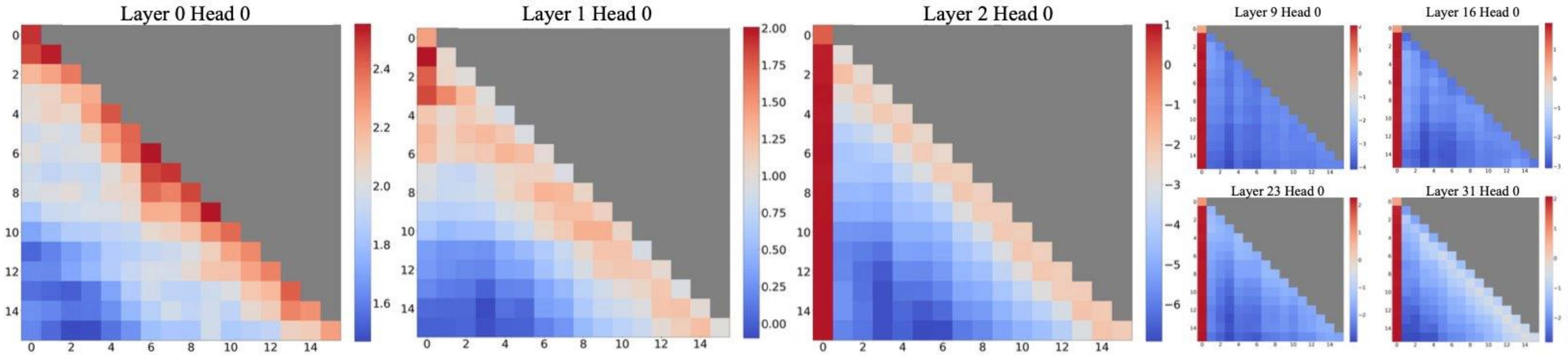


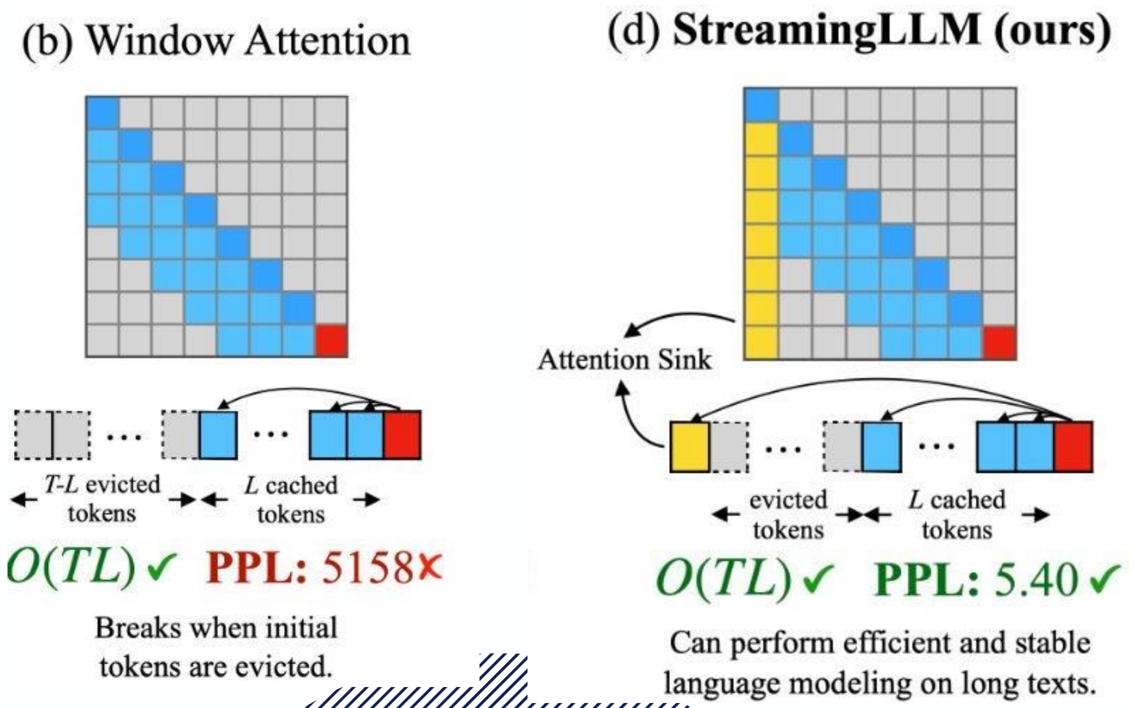
Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two layers, the model heavily attends to the initial token across all layers and heads.

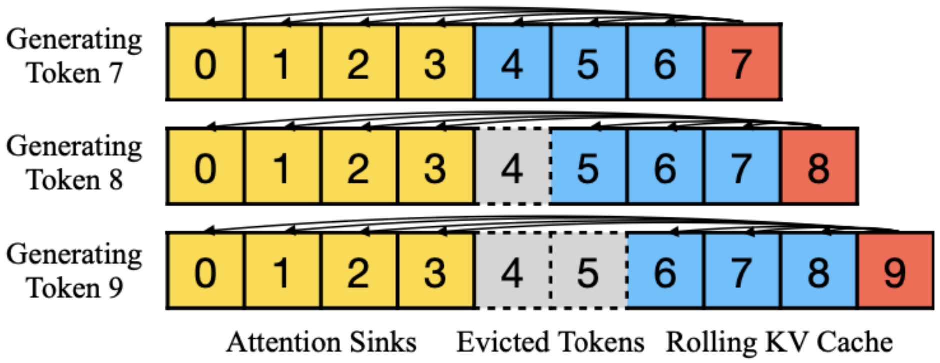
$$\mathrm{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^{N} e^{x_j}},$$

$$x_1 \gg x_j, j \in 2, \ldots, N$$

StreamingLLM

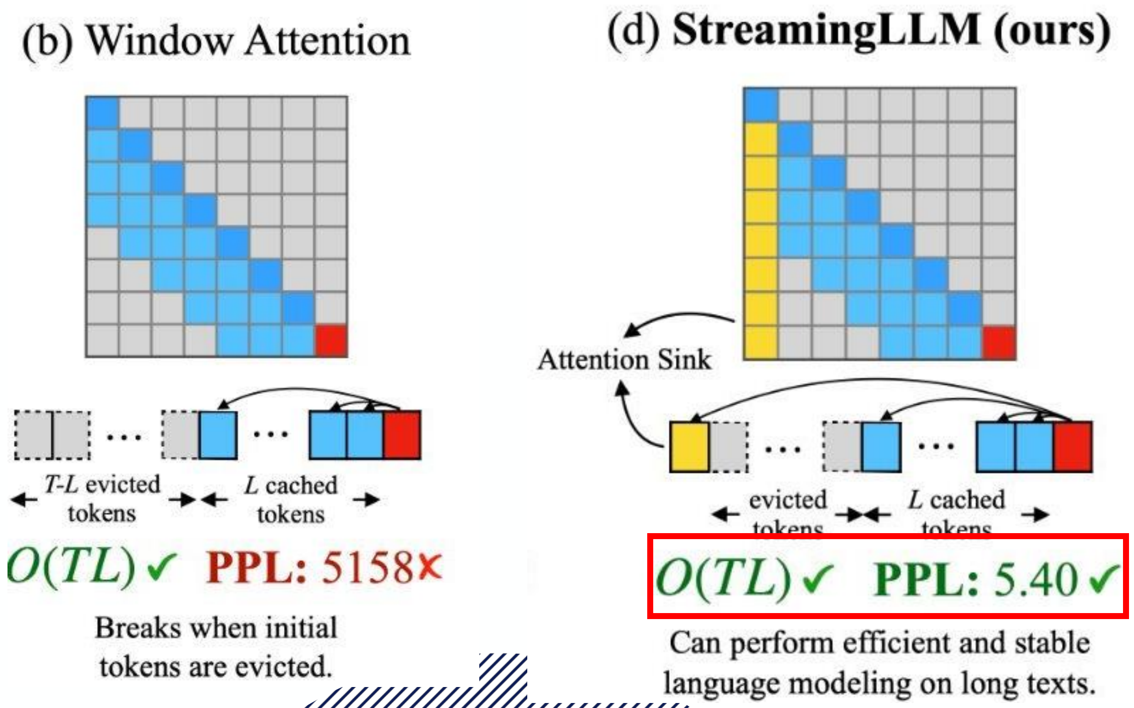
- **Goal:** Handle indefinite outputs without fine-tuning models
- **Method:** attention sinks + rolling KV cache

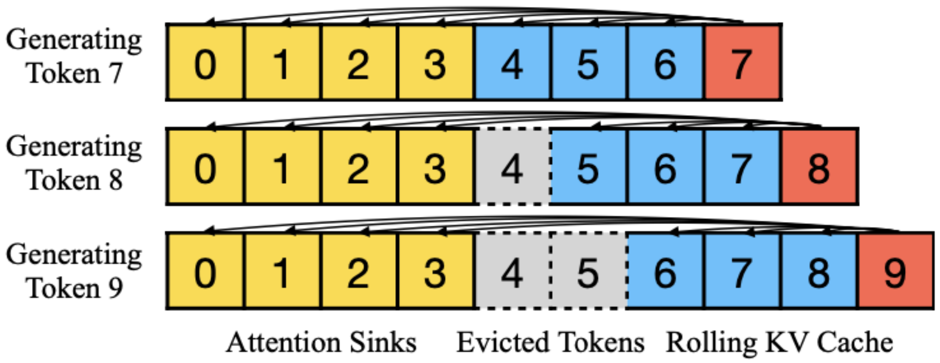




StreamingLLM

- **Goal:** Handle indefinite outputs without fine-tuning models
- **Method:** attention sinks + rolling KV cache





Experiments

How many attention sinks do we need?

Table 2: Effects of reintroduced initial token numbers on StreamingLLM. (1) Window attention (0+y) has a drastic increase in perplexity. (2) Introducing one or two initial tokens doesn't fully restore model perplexity, showing that the model doesn't solely use the first token as the attention sink. (3) Introducing four initial tokens generally suffices; further additions have diminishing returns. Cache config x+y denotes adding x initial tokens to y recent tokens. Perplexities are evaluated on 400K tokens in the concatenated PG19 test set.

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

How many attention sinks do we need?

Table 2: Effects of reintroduced initial token numbers on StreamingLLM. (1) Window attention (0+y) has a drastic increase in perplexity. (2) Introducing one or two initial tokens doesn't fully restore model perplexity, showing that the model doesn't solely use the first token as the attention sink. (3) Introducing four initial tokens generally suffices; further additions have diminishing returns. Cache config x+y denotes adding x initial tokens to y recent tokens. Perplexities are evaluated on 400K tokens in the concatenated PG19 test set.

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

Which is more important?

- **Position** or **Semantics**

Table 1: Window attention has poor performance on long text. The perplexity is restored when we reintroduce the initial four tokens alongside the recent 1020 tokens (4+1020). Substituting the original four initial tokens with linebreak tokens "\n" (4"\n"+1020) achieves comparable perplexity restoration. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexities are measured on the first book (65K tokens) in the PG19 test set.

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

Which is more important?

- **Position** or **Semantics**

Table 1: Window attention has poor performance on long text. The perplexity is restored when we reintroduce the initial four tokens alongside the recent 1020 tokens (4+1020). Substituting the original four initial tokens with linebreak tokens "\n" (4"\n"+1020) achieves comparable perplexity restoration. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexities are measured on the first book (65K tokens) in the PG19 test set.

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

Will attention sinks affect the model training?

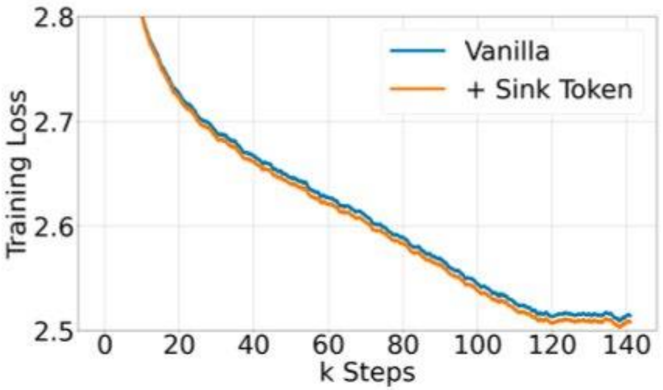


Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

Table 4: Zero-shot accuracy (in %) across 7 NLP benchmarks, including ARC-[Challenge, Easy], HellaSwag, LAMBADA, OpenbookQA, PIQA, and Winogrande. The inclusion of a sink token during pre-training doesn't harm the model performance.

Methods	ARC-c	ARC-e	HS	LBD	OBQA	PIQA	WG
Vanilla	18.6	45.2	29.4	39.6	16.0	62.2	50.1
+Sink Token	**19.6**	**45.6**	**29.8**	**39.9**	**16.6**	**62.6**	**50.8**

Will attention sinks affect the model training?

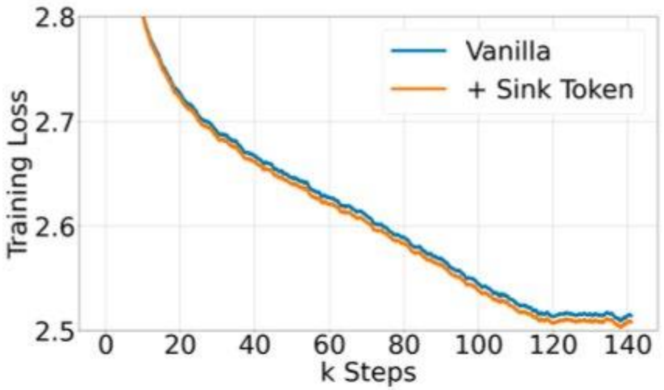


Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

Table 4: Zero-shot accuracy (in %) across 7 NLP benchmarks, including ARC-[Challenge, Easy], HellaSwag, LAMBADA, OpenbookQA, PIQA, and Winogrande. The inclusion of a sink token during pre-training doesn't harm the model performance.

Methods	ARC-c	ARC-e	HS	LBD	OBQA	PIQA	WG
Vanilla	18.6	45.2	29.4	39.6	16.0	62.2	50.1
+Sink Token	**19.6**	**45.6**	**29.8**	**39.9**	**16.6**	**62.6**	**50.8**

Efficiency

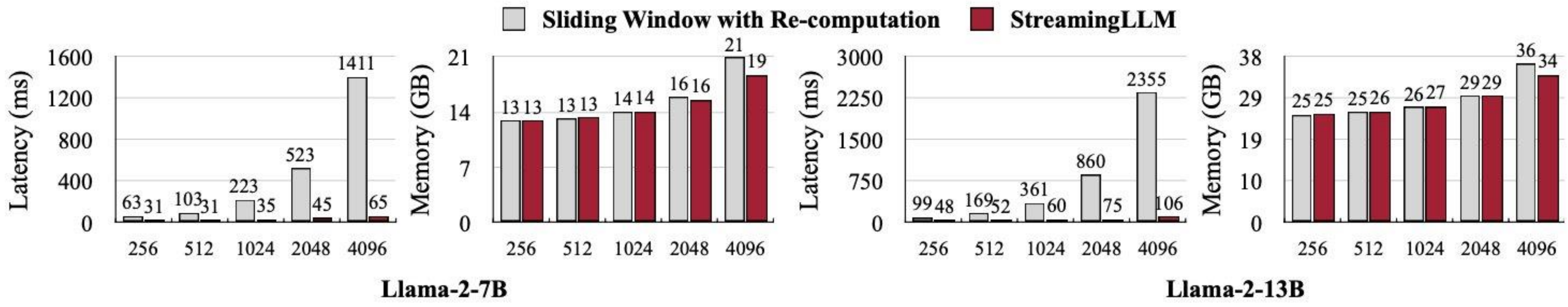


Figure 10: Comparison of per-token decoding latency and memory usage between the sliding window approach with re-computation baseline and StreamingLLM, plotted against the cache size (attention window size) on the X-axis. StreamingLLM delivers a remarkable speedup of up to 22.2× per token and retains a memory footprint similar to the re-computation baseline.

Thought

```python
from attention_sinks import AutoModel

model = AutoModel.from_pretrained("meta-llama/Llama-2-7b-hf", device_map="auto")
```

- **Application**:
  - Continuous Summaries: Provide a running summary of recent paragraphs or sections
  - Continuous conversational agents: Customer Support or Virtual Assistants
- **Strengths**:
  - Handle long/infinite sequence without fine-tuning
  - Comprehensive experimental investigations on different large language models
  - Easy to implement https://github.com/tomaarsen/attention_sinks

Thought (Cont'd)

- **Weaknesses**:
  - Based on empirical observation
  - Only autoregressive, decoder-only LMs, ex: GPT, Llama
- **Future Direction**:
  - Integrate with context-extension methods
  - Extend the work to different model architectures

THANK YOU!

ILLINOIS