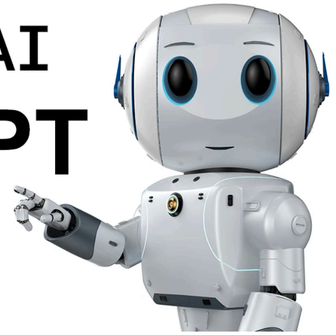
Efficient Streaming Language Models with Attention Sinks

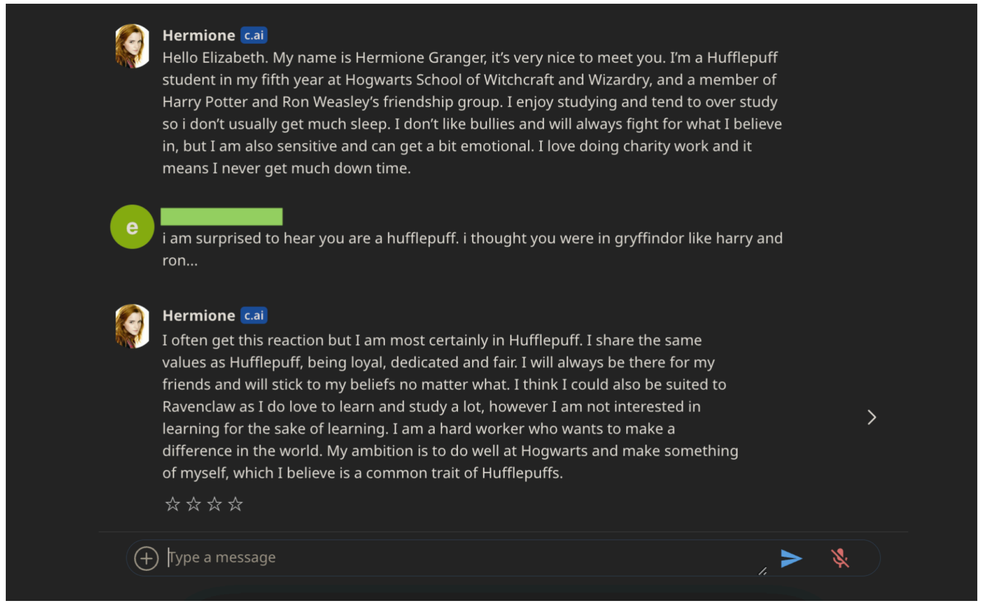
Guangxuan Xiao[1], Yuandong Tian[2], Beidi Chen[3], Song Han[1], Mike Lewis[2]

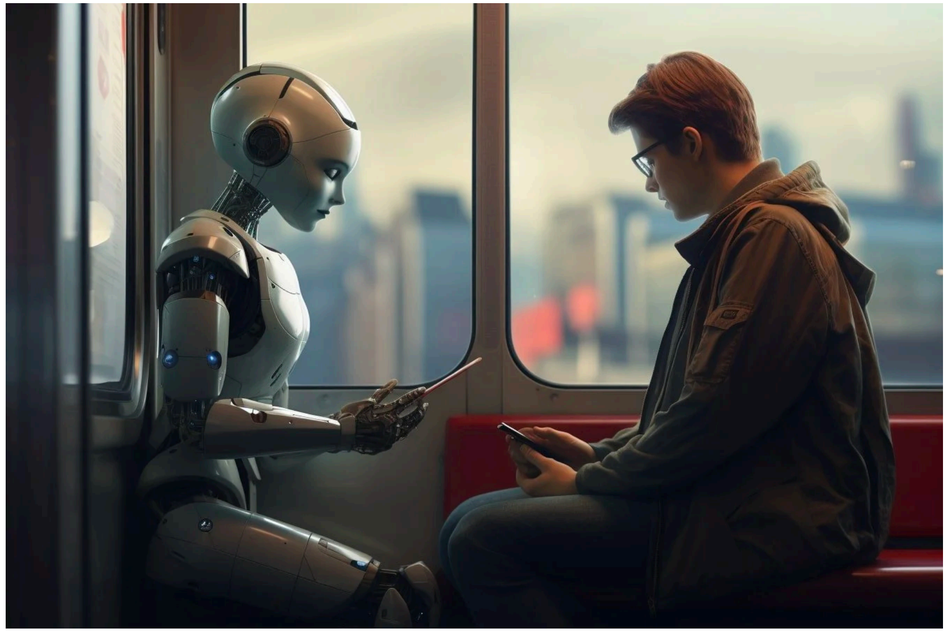
Massachusetts Institute of Technology[1]
Meta AI[2]
Carnegie Mellon University[3]

Motivation: Use cases

OpenAI
ChatGPT

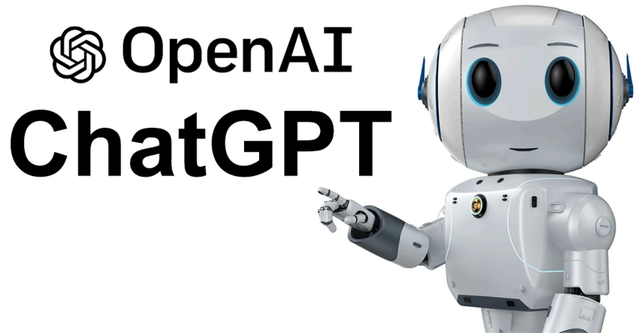




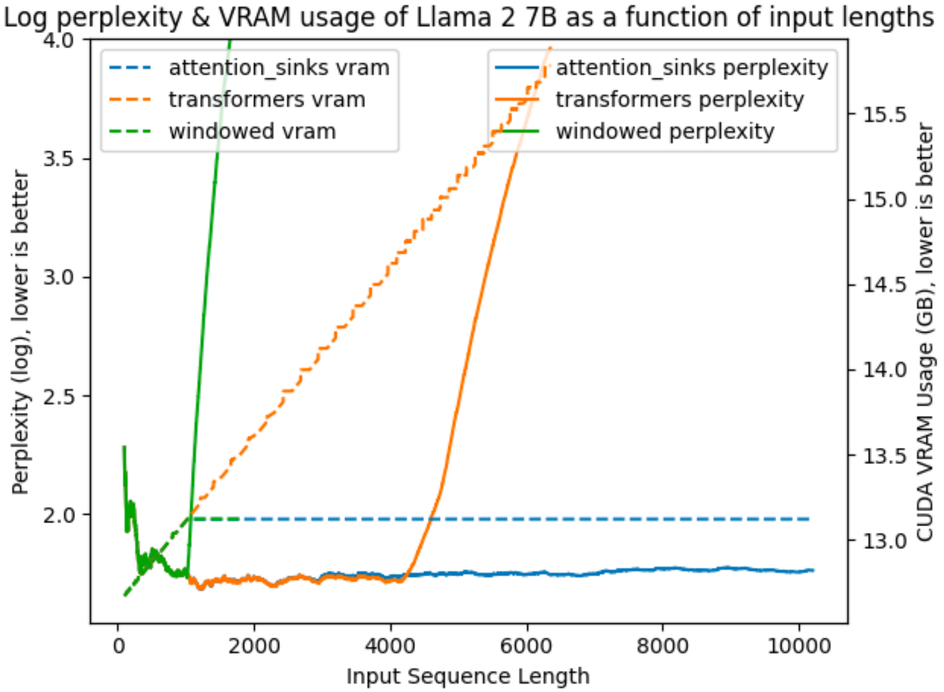


Challenges of Deploying LLMs in Streaming Applications

- Urgent need for LLMs in streaming applications such as multi-round dialogues, where long interactions are needed.

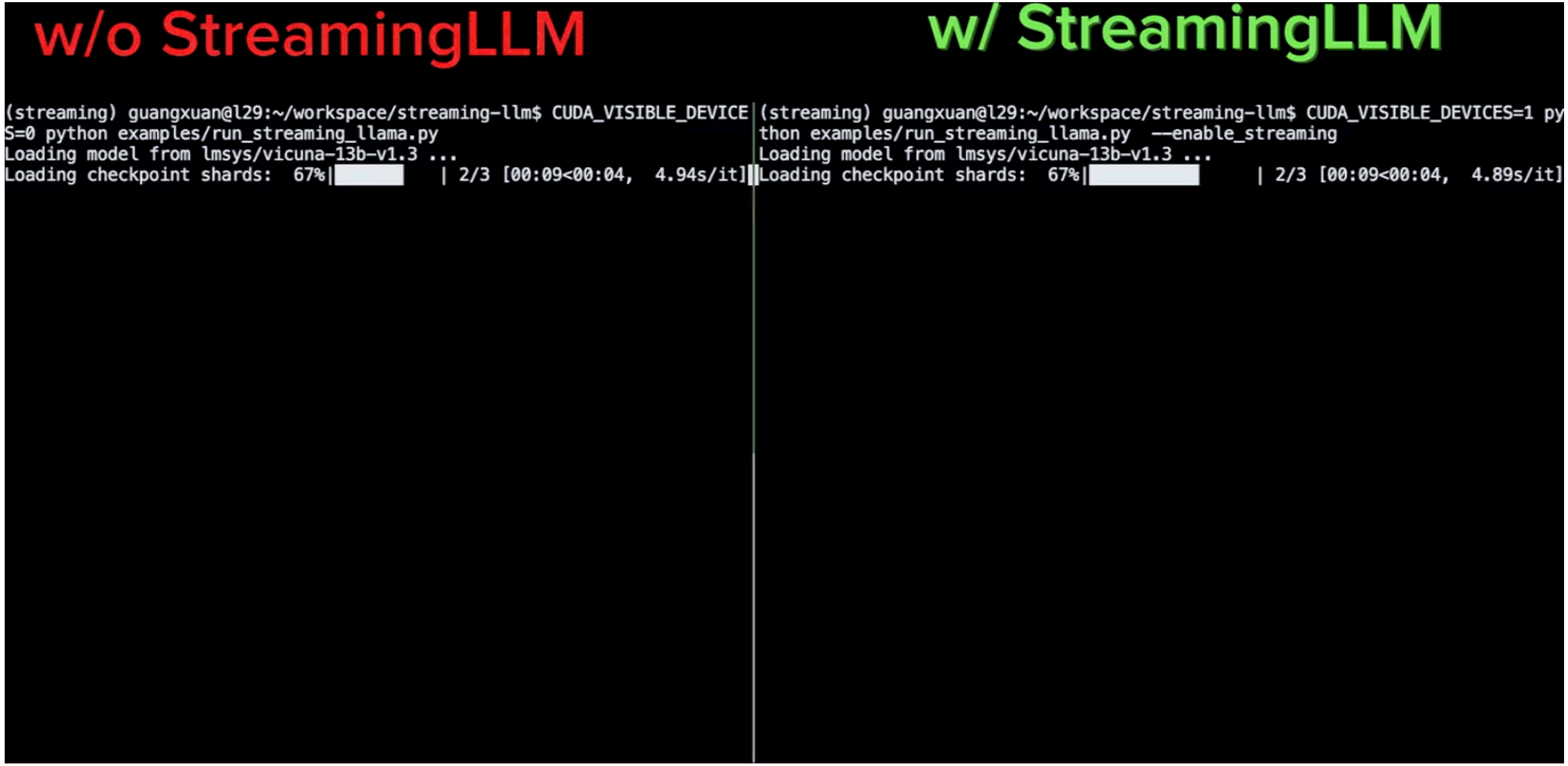


- Challenges:
  - Extensive memory consumption during the decoding stage.
  - Inability of popular LLMs to generalize to longer text sequences.

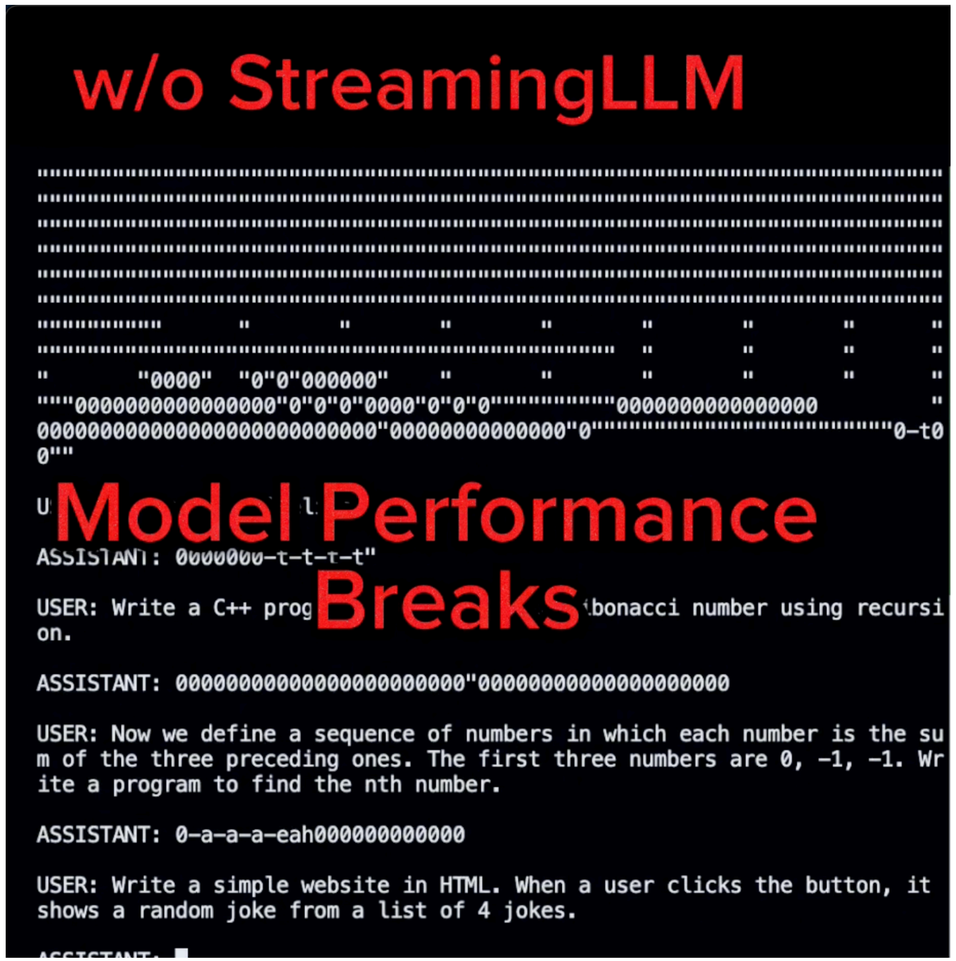


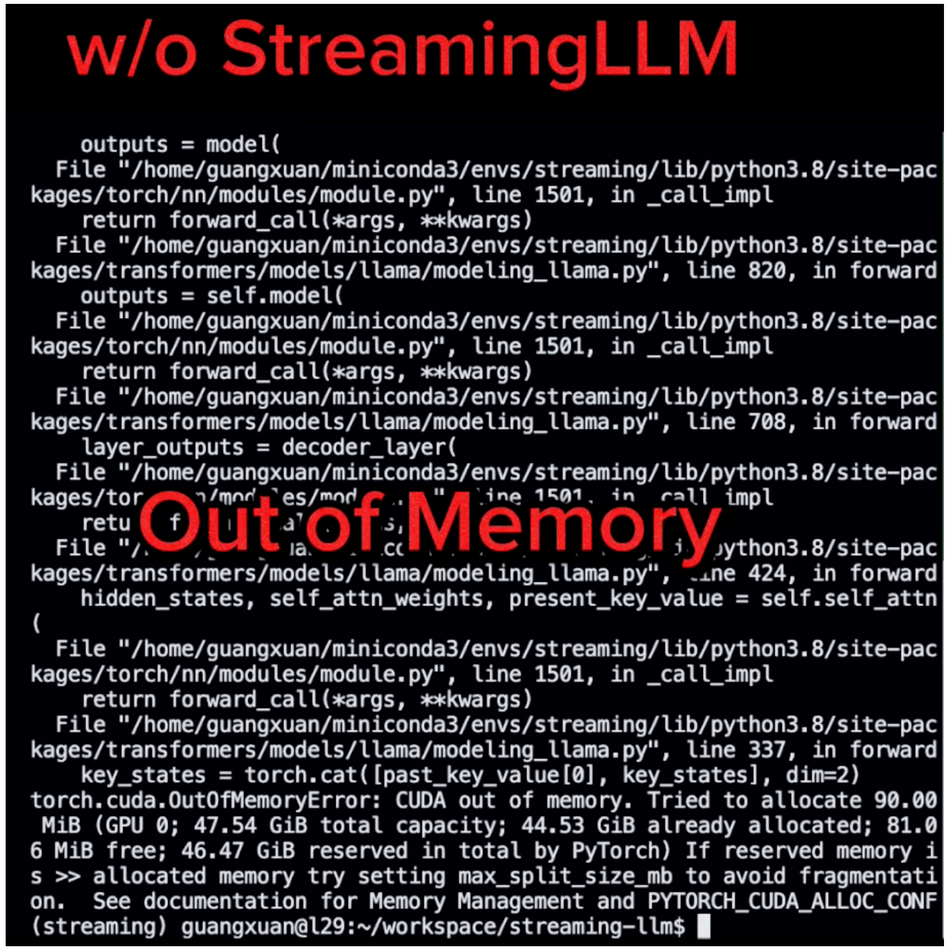
https://github.com/tomaarsen/attention_sinks

Challenges of Deploying LLMs in Streaming Applications



Challenges of Deploying LLMs in Streaming Applications

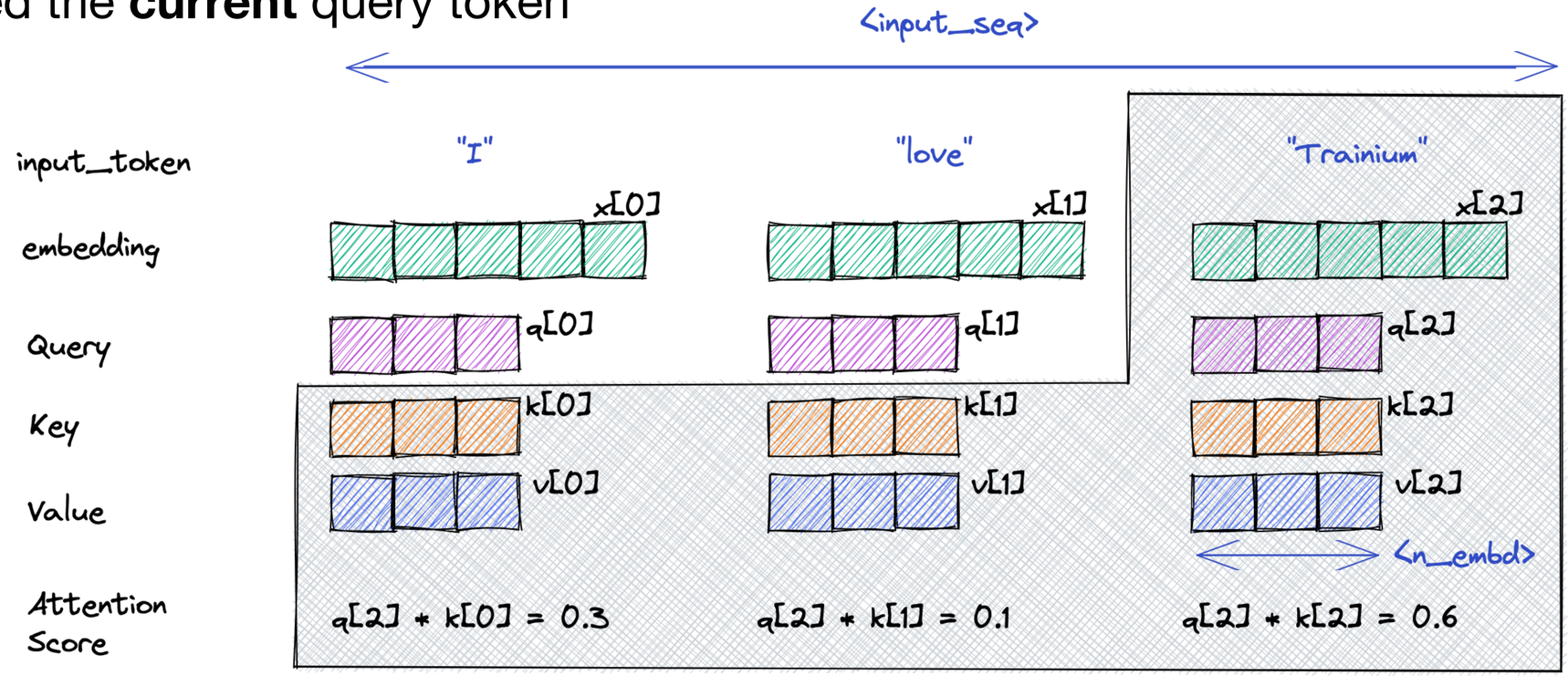




The Problem of Long Context: Large KV Cache

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **K**eys and **V**alues of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
  - Only need the **current** query token



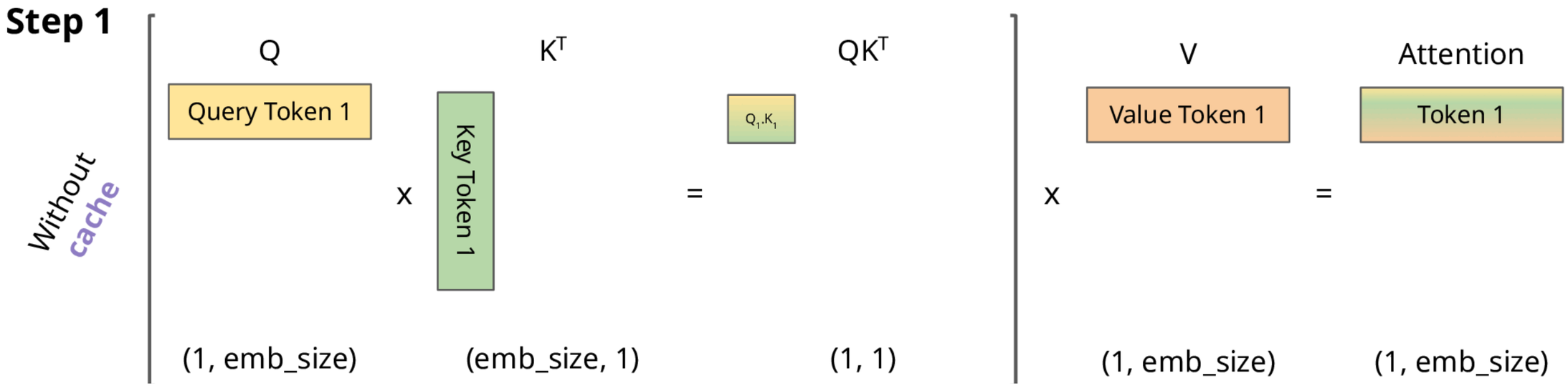
$$a_{ij} = \frac{\exp(q_i^\top k_j/\sqrt{d})}{\sum_{t=1}^{i}\exp(q_i^\top k_t/\sqrt{d})},\ o_i = \sum_{j=1}^{i} a_{ij} v_j$$

Image credit: https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/appnotes/transformers-neuronx/generative-llm-inference-with-neuron.html

The Problem of Long Context: Large KV Cache

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **K**eys and **V**alues of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
  - Only need the **current** query token



The Problem of Long Context: Large KV Cache

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **K**eys and **V**alues of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
  - Only need the **current** query token

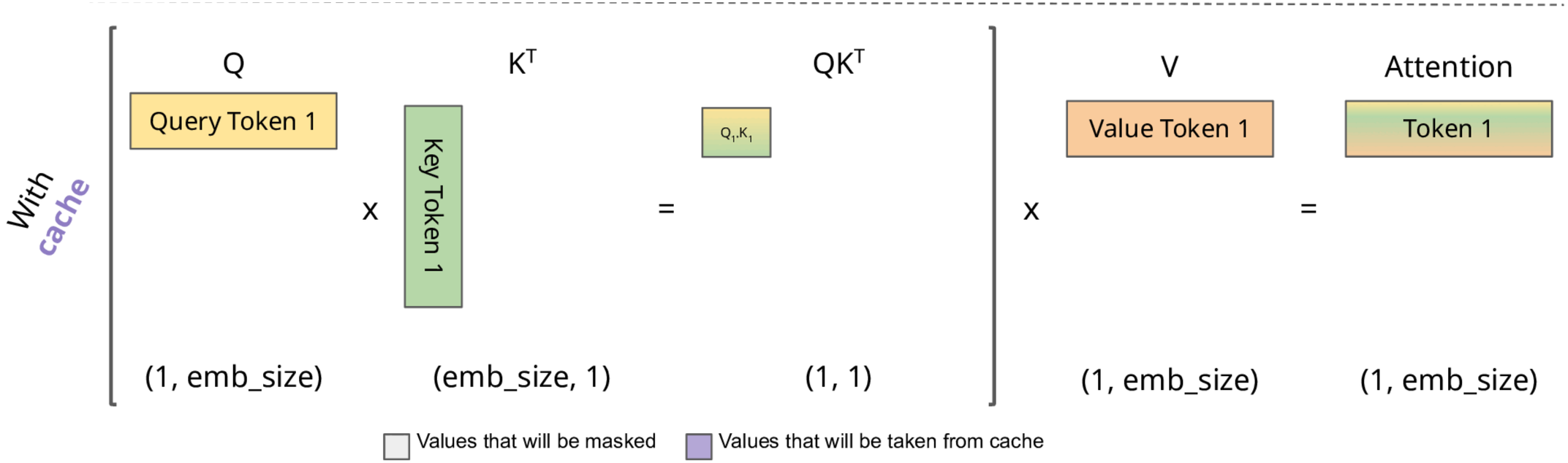


Image credit: https://medium.com/@joaolages/kv-caching-explained-276520203249

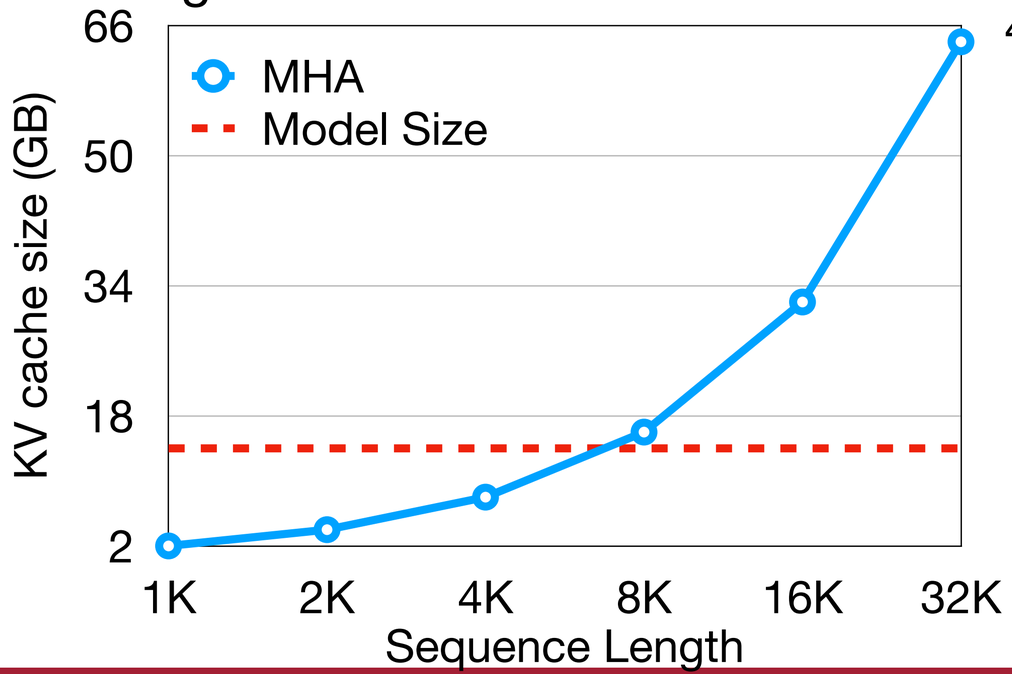
The Problem of Long Context: Large KV Cache

The KV cache could be large with long context

- We can calculate the memory required to store the KV cache
- Take Llama-2-7B as an example

$$\underbrace{BS}_{batchsize} * \underbrace{32}_{layers} * \underbrace{32}_{kv-heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \underbrace{2\text{bytes}}_{FP16} = 0.5\text{MB} \times BS \times N$$

- Now we calculate the KV cache size under $BS = 4$ and different sequence lengths.
  - Quickly larger than model weights

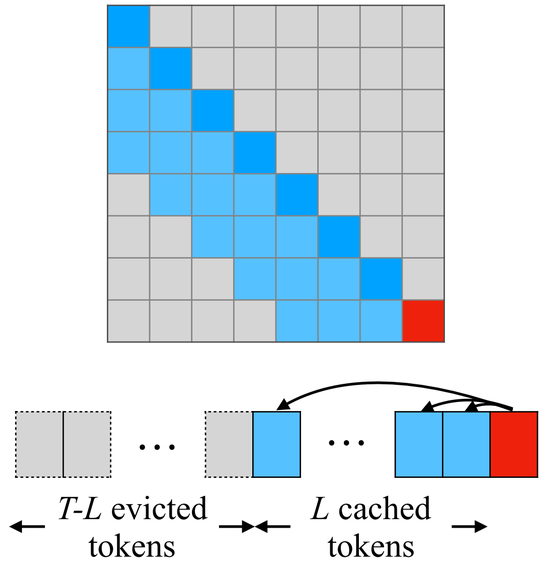


4*32*32*128*32K*2*2=**64GB**

The Limits of Window Attention

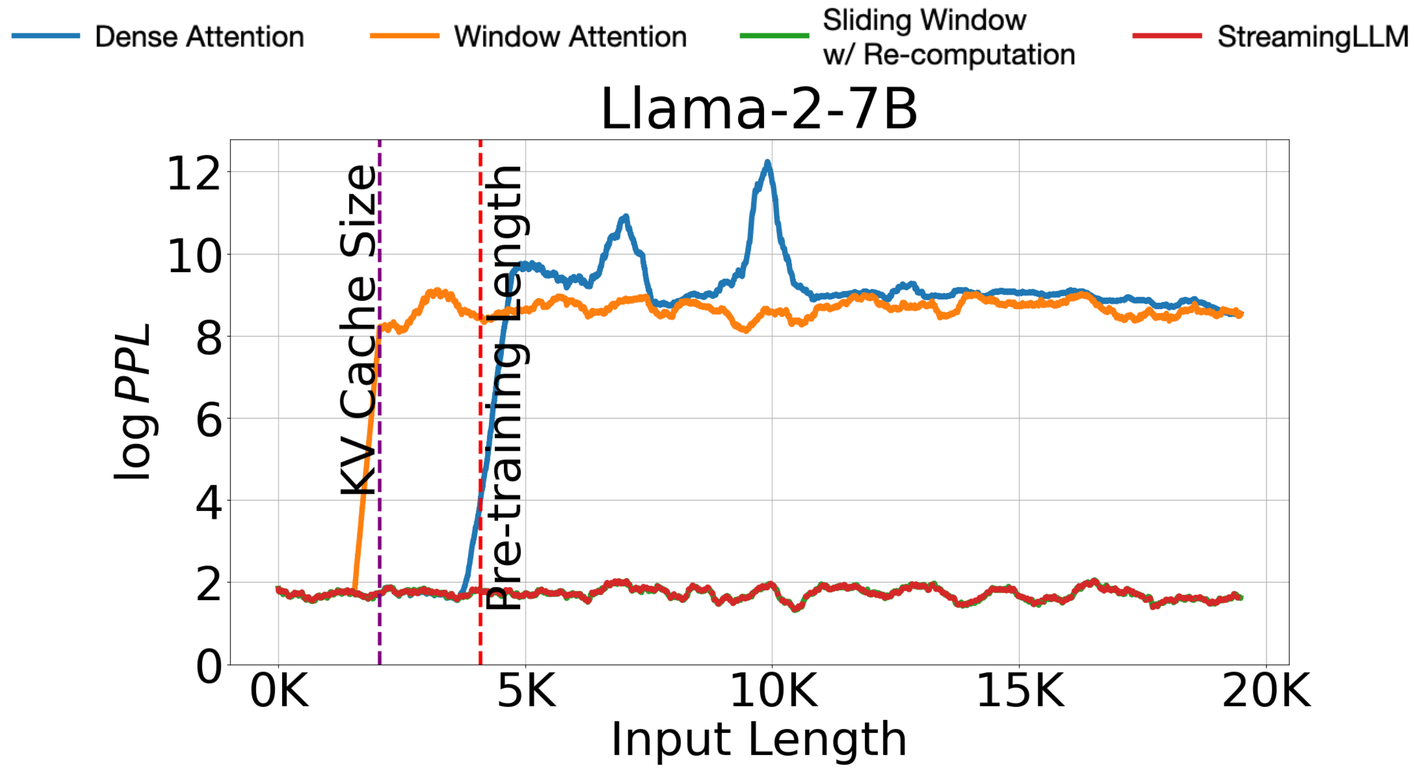
- A natural approach — window attention: caching only the most recent Key-Value states.
- Drawback: model collapses when the text length surpasses the cache size, when the initial token is evicted.

(b) Window Attention



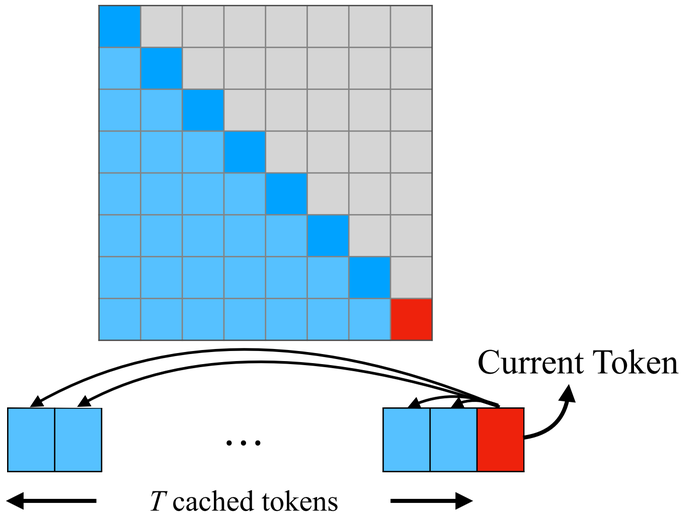
$O(TL)$ ✔ **PPL:** 5158

Breaks when initial tokens are evicted.



Difficulties of Other Methods

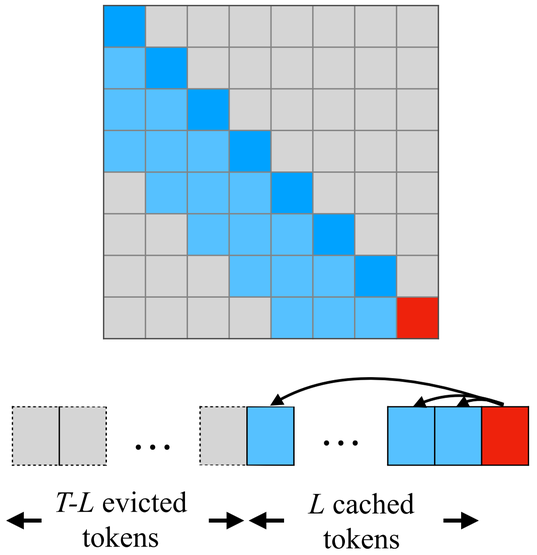
(a) Dense Attention



$O(T^2)$ ✗ **PPL:** 5641 ✗

Has poor efficiency and performance on long text.

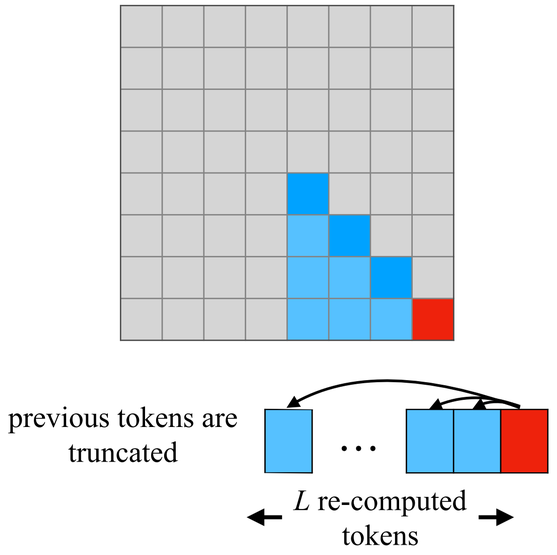
(b) Window Attention



$O(TL)$ ✓ **PPL:** 5158 ✗

Breaks when initial tokens are evicted.

(c) Sliding Window w/ Re-computation



$O(TL^2)$ ✗ **PPL:** 5.43 ✓

Has to re-compute cache for each incoming token.

The “Attention Sink” Phenomenon

- **Observation:** initial tokens have large attention scores, even if they're not semantically significant.
- **Attention Sink:** Tokens that disproportionately attract attention irrespective of their relevance.

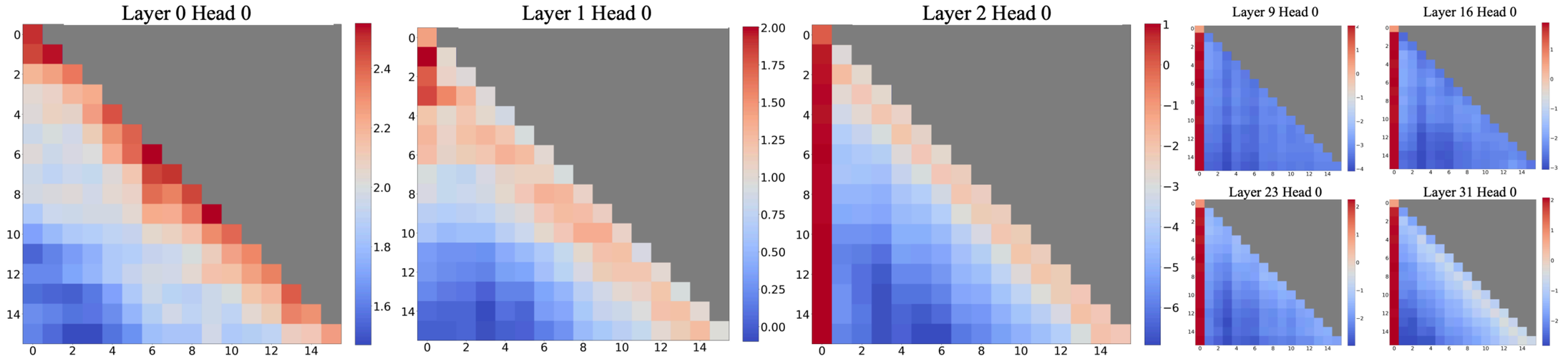


Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two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

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

The “Attention Sink” Phenomenon

- This phenomenon is observed in the SpAtten paper three years ago, but was not explored.

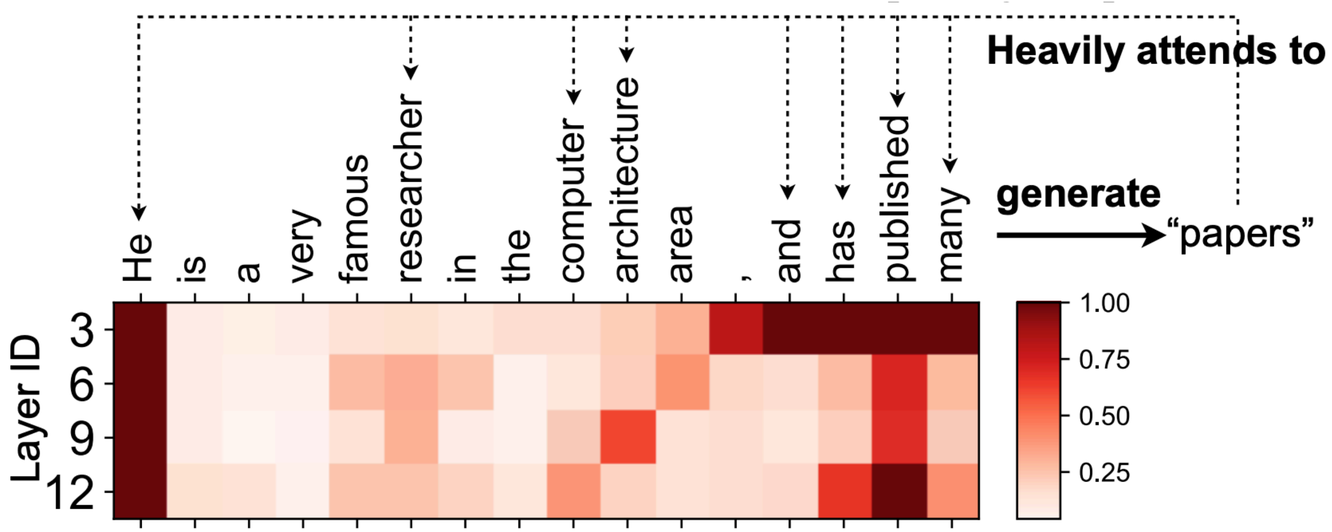


Fig. 23. Cumulative importance scores in GPT-2. Unimportant tokens are pruned on the fly. Important tokens are heavily attended.

GPT-2 for Language Modeling

Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into *English*
Du ~~Fu was a great~~ poet ~~of the~~ Tang dynasty. Recently ~~a variety of~~ styles ~~have been used~~ in efforts to translate the work of Du Fu into *English*
Du ~~Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to~~ translate ~~the work of Du Fu~~ into *English*

‘***English***’ is the generated token.

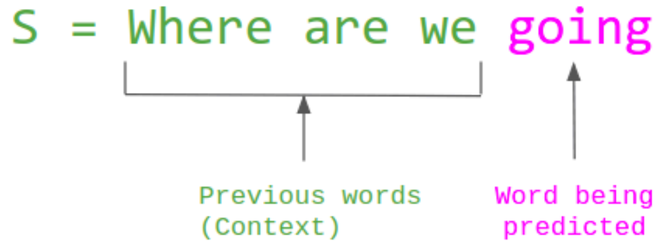
Understanding Why Attention Sinks Exist

The Rationale Behind Attention Sinks

- SoftMax operation's role in creating attention sinks — attention scores have to sum up to one for all contextual tokens.

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

- Initial tokens' advantage in becoming sinks due to their visibility to subsequent tokens, rooted in autoregressive language modeling.



P(S) = P(Where) x P(are | Where) x P(we | Where are) x P(going | Where are we)

- Does the importance of the initial tokens arise from their **position** or their **semantics**?
  - We found adding initial four “\n”s can also recover perplexity.
  - Therefore, it is **position**!

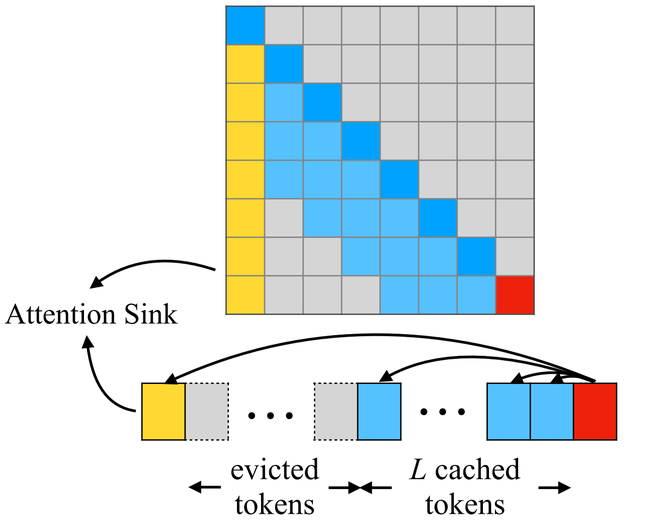
Table 1: Window attention has poor performance on long text. The perplexity is restored when we reintroduce the initial four tokens alongside the recent 1020 tokens (4+1020). Substituting the original four initial tokens with linebreak tokens “\n" (4"\n"+1020) achieves comparable perplexity restoration. 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

StreamingLLM: Using Attention Sinks for Infinite Streams

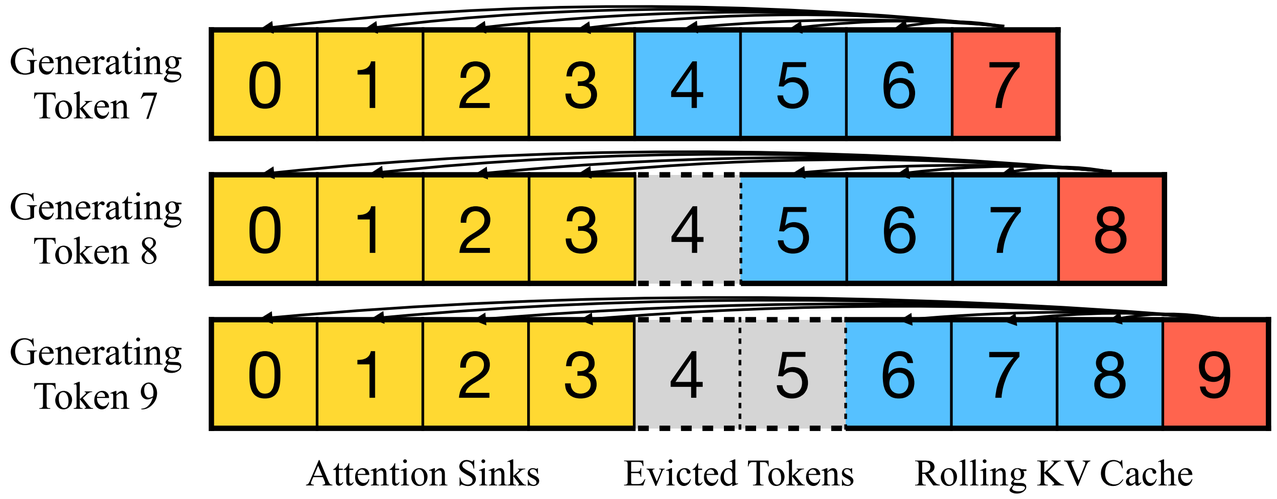
- **Objective:** Enable LLMs trained with a finite attention window to handle infinite text lengths without additional training.
- **Key Idea:** **preserve the KV of attention sink tokens**, along with the sliding window's KV to stabilize the model's behavior.

(d) **StreamingLLM (ours)**



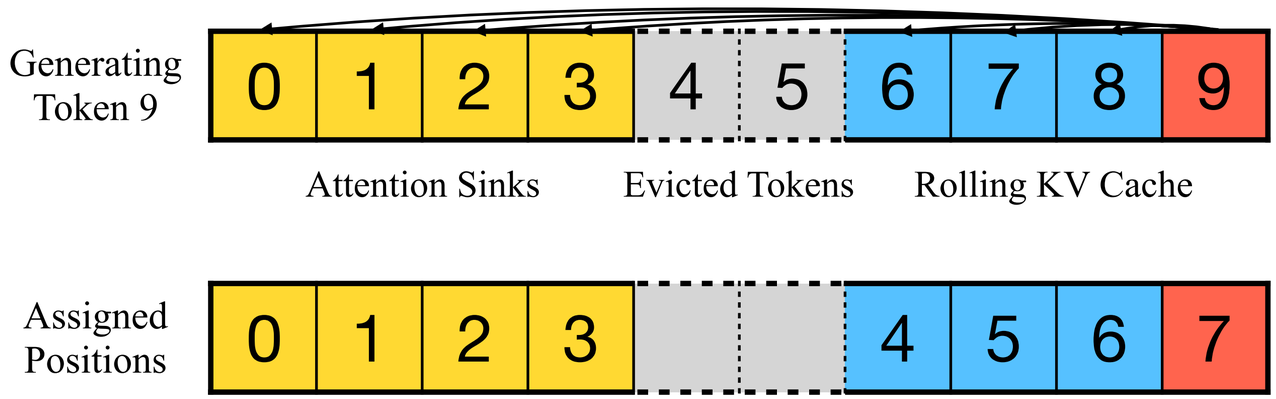
$O(TL)$ ✔ **PPL:** 5.40 ✔

Can perform efficient and stable language modeling on long texts.



Positional Encoding Assignment

- Use positions ***in the cache*** instead of those ***in the original text*.**



Streaming Performance

- Comparison between dense attention, window attention, and sliding window w/ re-computation.

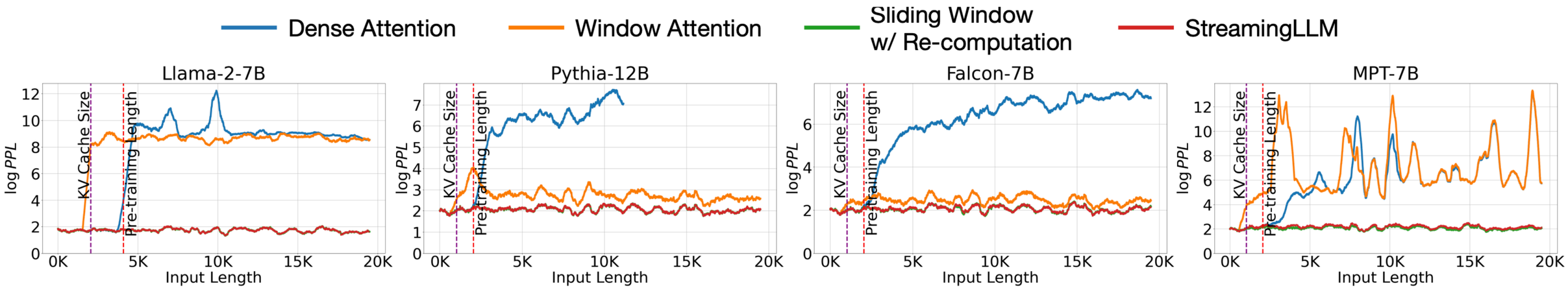


Figure 3: Language modeling perplexity on texts with 20K tokens across various LLM. Observations reveal consistent trends: (1) Dense attention fails once the input length surpasses the pre-training attention window size. (2) Window attention collapses once the input length exceeds the cache size, i.e., the initial tokens are evicted. (3) StreamingLLM demonstrates stable performance, with its perplexity nearly matching that of the sliding window with re-computation baseline.

Streaming Performance

Super Long Language Modeling

- With StreamingLLM, model families include Llama-2, MPT, Falcon, and Pythia can now effectively model up to 4 million tokens.

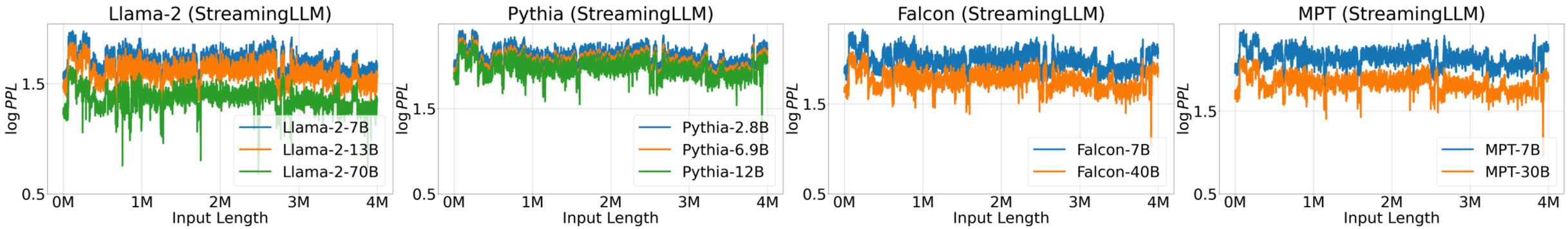
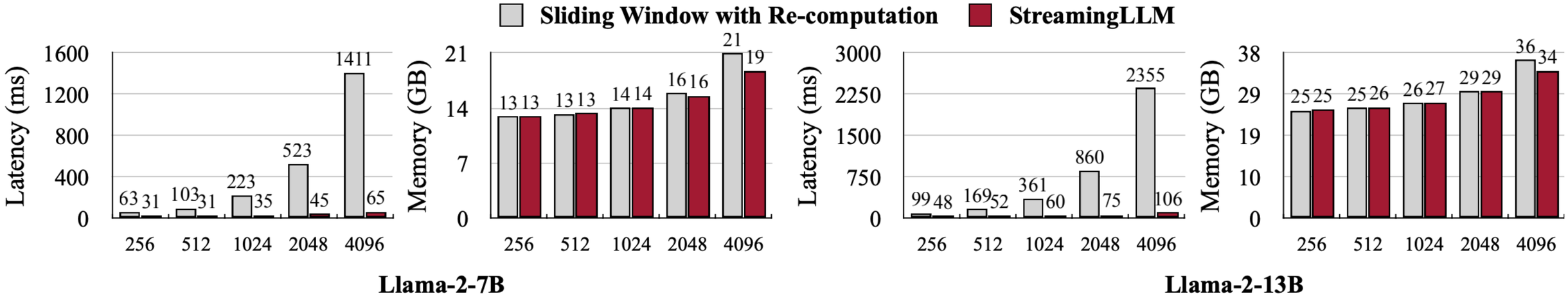


Figure 5: Language modeling perplexity of StreamingLLM for super long texts with 4 million tokens across various LLM families and model scales. The perplexity remains stable throughout. We use the concatenated test set of PG19 (100 books) to perform language modeling, with perplexity fluctuations attributable to the transition between books.

Efficiency

- **Comparison baseline:** The sliding window with re-computation, a method that is computationally heavy due to quadratic attention computation within its window.
- StreamingLLM provides up to 22.2x speedup over the baseline, making LLMs for real-time streaming applications feasible.



Ablation Study: #Attention Sinks

- The number of attention sinks that need to be introduced to recover perplexity.
  - 4 attention sinks are generally enough.

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 usually doesn't suffice to fully restore model perplexity, indicating 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

Pre-training with a Dedicated Attention Sink Token

- **Idea: Why 4 attention sinks?** Can we train a LLM that need only one single attention sink? **Yes!**
- **Method:** Introduce an extra learnable token at the start of all training samples to act as a dedicated attention sink.
- **Result:** This pre-trained model retains performance in streaming cases with just this single sink token, contrasting with vanilla models that require multiple initial tokens.

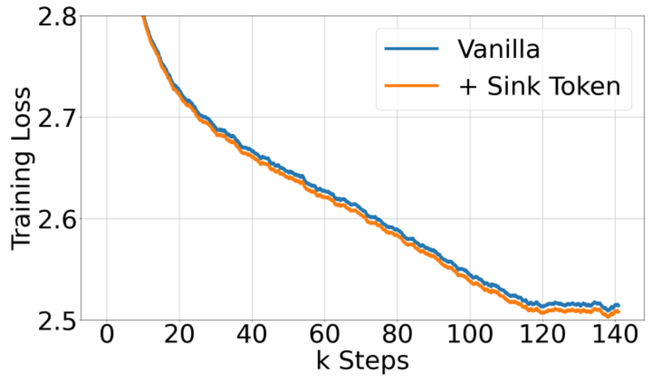


Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

Table 3: Comparison of vanilla attention with prepending a zero token and a learnable sink token during pre-training. To ensure stable streaming perplexity, the vanilla model required several initial tokens. While Zero Sink demonstrated a slight improvement, it still needed other initial tokens. Conversely, the model trained with a learnable Sink Token showed stable streaming perplexity with only the sink token added. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexity is evaluated on the first sample in the PG19 test set.

Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	**18.01**	18.01	18.02

Thanks for Listening!

- We propose StreamingLLM, enabling the streaming deployment of LLMs.

- Paper: https://arxiv.org/abs/2309.17453
- Code: https://github.com/mit-han-lab/streaming-llm
- Demo: https://youtu.be/UgDcZ3rvRPg