EFFICIENT STREAMING LANGUAGE MODELS WITH ATTENTION SINKS

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

[1] Massachusetts Institute of Technology
[2] Meta AI
[3] Carnegie Mellon University
https://github.com/mit-han-lab/streaming-llm

ABSTRACT

Deploying Large Language Models (LLMs) in streaming applications such as multi-round dialogue, where long interactions are expected, is urgently needed but poses two major challenges. Firstly, during the decoding stage, caching previous tokens' Key and Value states (KV) consumes extensive memory. Secondly, popular LLMs cannot generalize to longer texts than the training sequence length. Window attention, where only the most recent KVs are cached, is a natural approach — but we show that it fails when the text length surpasses the cache size. We observe an interesting phenomenon, namely *attention sink*, that keeping the KV of initial tokens will largely recover the performance of window attention. In this paper, we first demonstrate that the emergence of *attention sink* is due to the strong attention scores towards initial tokens as a "sink" even if they are not semantically important. Based on the above analysis, we introduce StreamingLLM, an efficient framework that enables LLMs trained with a *finite length* attention window to generalize to *infinite sequence length* without any fine-tuning. We show that StreamingLLM can enable Llama-2, MPT, Falcon, and Pythia to perform stable and efficient language modeling with up to 4 million tokens and more. In addition, we discover that adding a placeholder token as a dedicated attention sink during pre-training can further improve streaming deployment. In streaming settings, StreamingLLM outperforms the sliding window recomputation baseline by up to 22.2× speedup. Code and datasets are provided in the link.

1 INTRODUCTION

Large Language Models (LLMs) (Radford et al., 2018; Brown et al., 2020; Zhang et al., 2022; OpenAI, 2023; Touvron et al., 2023a;b) are becoming ubiquitous, powering many natural language processing applications such as dialog systems (Schulman et al., 2022; Taori et al., 2023; Chiang et al., 2023), document summarization (Goyal & Durrett, 2020; Zhang et al., 2023a), code completion (Chen et al., 2021; Rozière et al., 2023) and question answering (Kamalloo et al., 2023). To unleash the full potential of pretrained LLMs, they should be able to efficiently and accurately perform long sequence generation. For example, an ideal ChatBot assistant can stably work over the content of recent day-long conversations. However, it is very challenging for LLM to generalize to longer sequence lengths than they have been pretrained on, e.g., 4K for Llama-2 Touvron et al. (2023b).

The reason is that LLMs are constrained by the attention window during pre-training. Despite substantial efforts to expand this window size (Chen et al., 2023; kaiokendev, 2023; Peng et al., 2023) and improve training (Dao et al., 2022; Dao, 2023) and inference (Pope et al., 2022; Xiao et al., 2023; Anagnostidis et al., 2023; Zhang et al., 2023b) efficiency for lengthy inputs, the acceptable sequence length remains intrinsically *finite*, which doesn't allow persistent deployments.

In this paper, we first introduce the concept of LLM streaming applications and ask the question:

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

[*] Part of the work done during an internship at Meta AI.

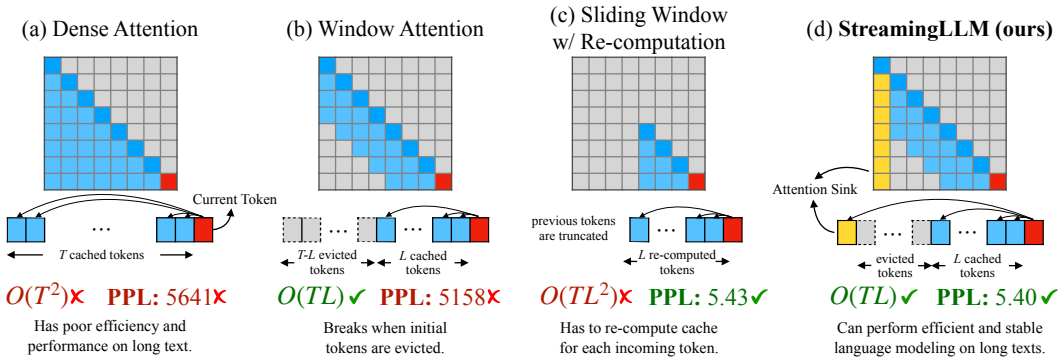

Figure 1: **Illustration of StreamingLLM *vs*. existing methods.** The language model, pre-trained on texts of length L, predicts the Tth token ($T \gg L$). (a) Dense Attention has $O(T^2)$ time complexity and an increasing cache size. Its performance decreases when the text length exceeds the pre-training text length. (b) Window Attention caches the most recent L tokens' KV. While efficient in inference, performance declines sharply once the starting tokens' keys and values are evicted. (c) Sliding Window with Re-computation rebuilds the KV states from the L recent tokens for each new token. While it performs well on long texts, its $O(TL^2)$ complexity, stemming from quadratic attention in context re-computation, makes it considerably slow. (d) StreamingLLM keeps the *attention sink* (several initial tokens) for stable attention computation, combined with the recent tokens. It's efficient and offers stable performance on extended texts. Perplexities are measured using the Llama-2-13B model on the first book (65K tokens) in the PG-19 test set.

When applying LLMs for infinite input streams, two primary challenges arise:

1. During the decoding stage, Transformer-based LLMs cache the Key and Value states (KV) of all previous tokens, as illustrated in Figure 1 (a), which can lead to excessive memory usage and increasing decoding latency (Pope et al., 2022).
2. Existing models have limited length extrapolation abilities, i.e., their performance degrades (Press et al., 2022; Chen et al., 2023) when the sequence length goes beyond the attention window size set during pre-training.

An intuitive approach, known as window attention (Beltagy et al., 2020) (Figure 1 b), maintains only a fixed-size sliding window on the KV states of most recent tokens. Although it ensures constant memory usage and decoding speed after the cache is initially filled, the model collapses once the sequence length exceeds the cache size, i.e., *even just evicting the KV of the first token*, as illustrated in Figure 3. Another strategy is the sliding window with re-computation (shown in Figure 1 c), which rebuilds the KV states of recent tokens for each generated token. While it offers strong performance, this approach is significantly slower due to the computation of quadratic attention within its window, making this method impractical for real-world streaming applications.

To understand the failure of window attention, we find an interesting phenomenon of autoregressive LLMs: a surprisingly large amount of attention score is allocated to the initial tokens, irrespective of their relevance to the language modeling task, as visualized in Figure 2. We term these tokens "**attention sinks**". Despite their lack of semantic significance, they collect significant attention scores. We attribute the reason to the Softmax operation, which requires attention scores to sum up to one for all contextual tokens. Thus, even when the current query does not have a strong match in many previous tokens, the model still needs to allocate these unneeded attention values somewhere so it sums up to one. The reason behind *initial* tokens as sink tokens is intuitive: initial tokens are visible to almost all subsequent tokens because of the autoregressive language modeling nature, making them more readily trained to serve as attention sinks.

Based on the above insights, we propose StreamingLLM, a simple and efficient framework that enables LLMs trained with a finite attention window to work on text of infinite length without fine-tuning. StreamingLLM exploits the fact that attention sinks have high attention values, and preserving them can maintain the attention score distribution close to normal. Therefore, StreamingLLM simply keeps the attention sink tokens' KV (with just 4 initial tokens sufficing) together with the sliding window's KV to anchor the attention computation and stabilize the model's performance. With StreamingLLM, models including Llama-2-[7, 13, 70]B, MPT-[7, 30]B, Falcon-[7, 40]B, and Pythia-

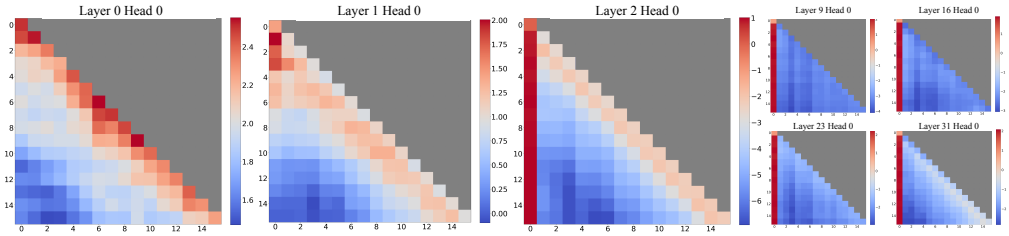

Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two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

[2.9,6.9,12]B can reliably model 4 million tokens, and potentially even more. Compared with the only viable baseline, sliding window with recomputation, StreamingLLM achieves up to 22.2× speedup, realizing the streaming use of LLMs.

Finally, we confirm our attention sink hypothesis and demonstrate that language models can be pre-trained to require only a single attention sink token for streaming deployment. Specifically, we suggest that an extra learnable token at the beginning of all training samples can serve as a designated attention sink. By pre-training 160-million parameter language models from scratch, we demonstrate that adding this single sink token preserves the model's performance in streaming cases. This stands in contrast to vanilla models, which necessitate the reintroduction of multiple initial tokens as attention sinks to achieve the same performance level.

2 Related Work

Extensive research has been done on applying LLMs to lengthy texts, with three main areas of focus: **Length Extrapolation**, **Context Window Extension**, and **Improving LLMs' Utilization of Long Text**. While seemingly related, it's worth noting that progress in one direction doesn't necessarily lead to progress in the other. For example, extending the context size of LLMs doesn't improve the model's performance beyond the context size, and neither approach ensures effective use of the long context. Our StreamingLLM framework primarily lies in the first category, where LLMs are applied to text significantly exceeding the pre-training window size, potentially even of infinite length. We do not expand the attention window size of LLMs or enhance the model's memory and usage on long texts. The last two categories are orthogonal to our focus and could be integrated with our techniques.

Length extrapolation aims to enable language models trained on shorter texts to handle longer ones during testing. A predominant avenue of research targets the development of relative position encoding methods for Transformer models, enabling them to function beyond their training window. One such initiative is Rotary Position Embeddings (RoPE) (Su et al., 2021), which transforms the queries and keys in every attention layer for relative position integration. Despite its promise, subsequent research (Press et al., 2022; Chen et al., 2023) indicated its underperformance on text that exceeds the training window. Another approach, ALiBi (Press et al., 2022), biases the query-key attention scores based on their distance, thereby introducing relative positional information. While this exhibited improved extrapolation, our tests on MPT models highlighted a breakdown when the text length was vastly greater than the training length. Current methodologies, however, have yet to achieve infinite length extrapolation, causing no existing LLMs to fit for streaming applications.

Context Window Extension centers on expanding the LLMs' context window, enabling the processing of more tokens in one forward pass. A primary line of work addresses the training efficiency problem. Given the attention to computation's quadratic complexity during training, developing a long-context LLM is both a computational and memory challenge. Solutions have ranged from system-focused optimizations like FlashAttention (Dao et al., 2022; Dao, 2023), which accelerates attention computation and reduces memory footprint, to approximate attention methods (Zaheer et al., 2020; Beltagy et al., 2020; Wang et al., 2020; Kitaev et al., 2020) that trade model quality for efficiency. Recently, there has been a surge of work on extending pre-trained LLMs with RoPE (Chen et al., 2023; kaiokendev, 2023; bloc97, 2023; Peng et al., 2023), involving position interpolation and fine-tuning. However, all the aforementioned techniques only extend LLMs' context window to a limited extent, which falls short of our paper's primary concern of handling limitless inputs.

Improving LLMs' Utilization of Long Text optimizes LLMs to better capture and employ the content within the context rather than merely taking them as inputs. As highlighted by Liu et al.

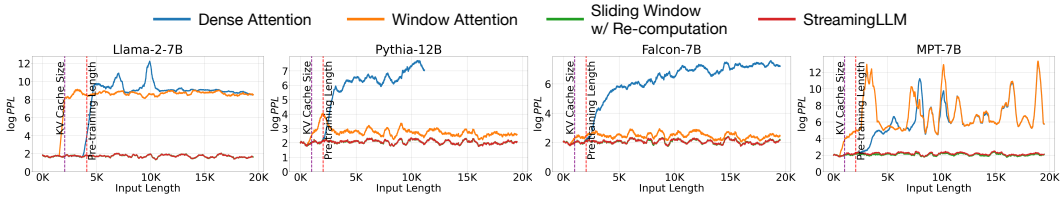

Figure 3: Language modeling perplexity on texts with 20K tokens across various LLM. Observations reveal consistent trends: (1) Dense attention fails once the input length surpasses the pre-training attention window size. (2) Window attention collapses once the input length exceeds the cache size, i.e.,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

and Li et al., success in the previously mentioned two directions does not necessarily translate to competent utilization of lengthy contexts. Addressing this effective usage of prolonged contexts within LLMs is still a challenge. Our work concentrates on stably harnessing the most recent tokens, enabling the seamless streaming application of LLMs.

3 StreamingLLM

3.1 The Failure of Window Attention and Attention Sinks

While the window attention technique offers efficiency during inference, it results in an exceedingly high language modeling perplexity. Consequently, the model's performance is unsuitable for deployment in streaming applications. In this section, we use the concept of *attention sink* to explain the failure of window attention, serving as the inspiration behind StreamingLLM.

Identifying the Point of Perplexity Surge. Figure 3 shows the perplexity of language modeling on a 20K token text. It is evident that perplexity spikes when the text length surpasses the cache size, led by the exclusion of initial tokens. This suggests that the initial tokens, regardless of their distance from the tokens being predicted, are crucial for maintaining the stability of LLMs.

Why do LLMs break when removing *initial* tokens' KV? We visualize attention maps from all layers and heads of the Llama-2-7B and models in Figure 2. We find that, beyond the bottom two layers, the model consistently focuses on the initial tokens across all layers and heads. The implication is clear: removing these initial tokens' KV will remove a considerable portion of the denominator in the SoftMax function (Equation 1) in attention computation. This alteration leads to a significant shift in the distribution of attention scores away from what would be expected in normal inference settings.

$$\text{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^{N} e^{x_j}}, \quad x_1 \gg x_j, j \in 2, \ldots, N \tag{1}$$

There are two possible explanations for the importance of the initial tokens in language modeling: (1) Either their semantics are crucial, or (2) the model learns a bias towards their absolute position. To distinguish between these possibilities, we conduct experiments (Table 1), wherein the first four tokens are substituted with the linebreak token "\n". The observations indicate that the model still significantly emphasizes these initial linebreak tokens. Furthermore, reintroducing them restores the language modeling perplexity to levels comparable to having the original initial tokens. This suggests that the absolute position of the starting tokens, rather than their semantic value, holds greater significance.

LLMs attend to Initial Tokens as Attention Sinks. To explain why the model disproportionately focuses on initial tokens—regardless of their semantic relevance to language modeling, we introduce the concept of "*attention sink*". The nature of the SoftMax function (Equation 1) prevents all attended tokens from having zero values. This requires aggregating some information from other tokens across all heads in all layers, even if the current embedding has sufficient self-contained information for its prediction. Consequently, the model tends to dump unnecessary attention values to specific tokens. A similar observation has been made in the realm of quantization outliers (Xiao et al., 2023; Bondarenko et al., 2023), leading to the proposal of SoftMax-Off-by-One (Miller, 2023) as a potential remedy.

Not sure how true this statement is. It seems like models tend to just stuff important information in unimportant tokens. This would align more with observation of quantization outliers

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

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

Why do various autoregressive LLMs, such as Llama-2, MPT, Falcon, and Pythia, consistently focus on *initial tokens* as their attention sinks, rather than other tokens? Our explanation is straightforward: Due to the sequential nature of autoregressive language modeling, initial tokens are visible to all subsequent tokens, while later tokens are only visible to a limited set of subsequent tokens. As a result, initial tokens are more easily trained to serve as attention sinks, capturing unnecessary attention.

We've noted that LLMs are typically trained to utilize multiple initial tokens as attention sinks rather than just one. As illustrated in Figure 2, the introduction of four initial tokens, as attention sinks, suffices to restore the LLM's performance. In contrast, adding just one or two doesn't achieve full recovery. We believe this pattern emerges because these models didn't include a consistent starting token across all input samples during pre-training. Although Llama-2 does prefix each paragraph with a "<s>" token, it's applied before text chunking, resulting in a mostly random token occupying the zeroth position. This lack of a uniform starting token leads the model to use several initial tokens as attention sinks. We hypothesize that by incorporating a stable learnable token at the start of all training samples, it could singularly act as a committed attention sink, eliminating the need for multiple initial tokens to ensure consistent streaming. We will validate this hypothesis in Section 3.3.

3.2 Rolling KV Cache with Attention Sinks

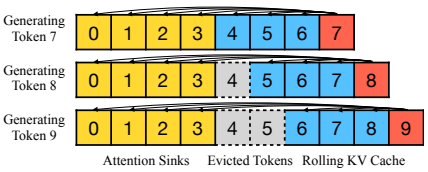

Figure 4: The KV cache of StreamingLLM.

To enable LLM streaming in already trained LLMs, we propose a straightforward method that can recover window attention's perplexity without any model finetuning. Alongside the current sliding window tokens, we reintroduce a few starting tokens' KV in the attention computation. The KV cache in StreamingLLM can be conceptually divided into two parts, as illustrated in Figure 4: (1) Attention sinks (four initial tokens) stabilize the attention computation; 2) Rolling KV Cache retains the most recent tokens, crucial for language modeling. StreamingLLM' design is versatile and can be seamlessly incorporated into any autoregressive language model that employs relative positional encoding, such as RoPE (Su et al., 2021) and ALiBi (Press et al., 2022).

When determining the relative distance and adding positional information to tokens, StreamingLLM focuses on positions *within the cache* rather than those *in the original text*. This distinction is crucial for StreamingLLM's performance. For instance, if the current cache has tokens [0, 1, 2, 3, 6, 7, 8] and is in the process of decoding the 9th token, the positions assigned are [0, 1, 2, 3, 4, 5, 6, 7], rather than the positions in the original text, which would be [0, 1, 2, 3, 6, 7, 8, 9].

For encoding like RoPE, we cache the Keys of tokens *prior to* introducing the rotary transformation. Then, we apply position transformation to the keys in the rolling cache at each decoding phase. On the other hand, integrating with ALiBi is more direct. Here, the contiguous linear bias is applied instead of a 'jumping' bias to the attention scores. This method of assigning positional embedding within the cache is crucial to StreamingLLM's functionality, ensuring that the model operates efficiently even beyond its pre-training attention window size.

3.3 Pre-Training LLMs with Attention Sinks

As elaborated in Section 3.1, a significant reason for the model's excessive attention to multiple initial tokens is the absence of a designated sink token to offload excessive attention scores. Due to this, the model inadvertently designates globally visible tokens, primarily the initial ones, as attention sinks. A potential remedy can be the intentional inclusion of a global trainable attention sink token, denoted as a "Sink Token", which would serve as a repository for unnecessary attention scores. Alternatively, replacing the conventional SoftMax function with a variant like SoftMax-off-by-One (Miller, 2023),

This additional 1 is to allow the model to "not-learn" for information it deems useless

$$\text{SoftMax}_1(x)_i = \frac{e^{x_i}}{1 + \sum_{j=1}^{N} e^{x_j}}, \qquad (2)$$

which does not require the attention scores on all contextual tokens to sum up to one, might also be effective. Note that this SoftMax alternative is equivalent to using a token with an all-zero Key and Value features in the attention computation. We denote this method as "Zero Sink" to fit it consistently in our framework.

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

For validation, we pre-train three language models with 160 million parameters from scratch under identical settings. The first model utilizes the standard SoftMax attention (Vanilla), the second replaced the regular attention mechanism with SoftMax_1 (Zero Sink), and one prepending a learnable placeholder token (Sink Token) in all training samples. As shown in Table 3, while the zero sink alleviates the attention sink problem to some extent, the model still relies on other initial tokens as attention sinks. Introducing a sink token is highly effective in stabilizing the attention mechanism. Simply pairing this sink token with recent tokens sufficiently anchors the model's performance, and the resulting evaluation perplexity is even marginally improved. Given these findings, we recommend training future LLMs with a sink token in all samples to optimize streaming deployment.

4 Experiments

We evaluate StreamingLLM using four prominent recent model families: Llama-2 (Touvron et al., 2023b), MPT (Team, 2023), PyThia (Biderman et al., 2023), and Falcon (Almazrouei et al., 2023). Notably, Llama-2, Falcon, and Pythia incorporate RoPE (Su et al., 2021), whereas MPT employs ALiBi (Press et al., 2022) — two of the most influential position encoding techniques in recent research. Our diverse model selection ensures the validity and robustness of our findings. We benchmark StreamingLLM against established baselines such as dense attention, window attention, and the sliding window approach with re-computation. In all subsequent experiments with StreamingLLM, we default to using four initial tokens as attention sinks unless stated otherwise.

4.1 Language Modeling on Long Texts Across LLM Families and Scales

We firstly evaluate StreamingLLM's language modeling perplexity using the concatenated PG19 (Rae et al., 2020) test set, which contains 100 long books. For Llama-2 models, the cache size is set at 2048, while for Falcon, Pythia, and MPT models, it's set at 1024. This is half the pre-training window size chosen to enhance visualization clarity.

Figure 3 illustrates that StreamingLLM can match the oracle baseline (sliding window with re-computation) in terms of perplexity on texts spanning 20K tokens. Meanwhile, the dense attention technique fails when the input length exceeds its pre-training window, and the window attention technique struggles when the input length surpasses the cache size, leading to the eviction of the initial tokens. In Figure 5, we further substantiate that StreamingLLM can reliably handle exceptionally extended texts, encompassing more than 4 million tokens, across a spectrum of model families and scales. This includes Llama-2-[7,13,70]B, Falcon-[7,40]B, Pythia-[2.8,6.9,12]B, and MPT-[7,30]B.

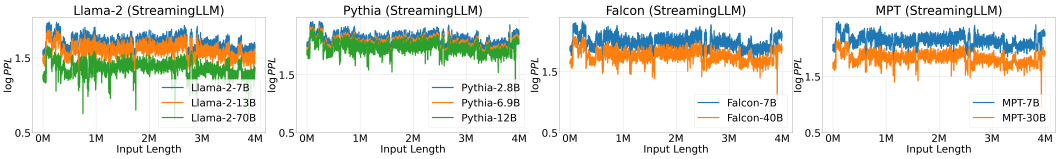

Figure 5: Language modeling perplexity of StreamingLLM for super long texts with 4 million tokens across various LLM families and model scales. The perplexity remains stable throughout. We use the concatenated test set of PG19 (100 books) to perform language modeling, with perplexity fluctuations attributable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

4.2 Results of Pre-Training with a Sink Token

To validate our suggestion that introducing a sink token to all pre-training samples improves streaming LLMs, we trained two language models, each with 160 million parameters, under identical conditions. While one model adhered to the original training settings, the other incorporated a sink token at the start of every training sample. Our experiments employed the Pythia-160M (Biderman et al., 2023) codebase and followed its training recipe. We train the models on an 8xA6000 NVIDIA GPU server using the deduplicated Pile (Gao et al., 2020) dataset. Apart from reducing the training batch size to 256, we retained all Pythia training configurations, including learning rate schedules, model initialization, and dataset permutations. Both models were trained for 143,000 steps.

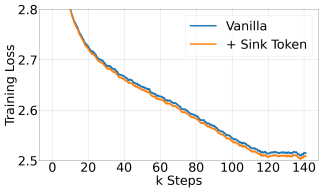

Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

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**

Convergence and Normal Model Performance. Including a sink token during pre-training has no negative impact on model convergence and subsequent performance on a range of NLP benchmarks. As depicted in Figure 6, models trained with a sink token exhibit similar convergence dynamics compared to their vanilla counterparts. We evaluate the two models on seven diverse NLP benchmarks, including ARC-[Challenge, Easy] (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), OpenbookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and Winogrande (Sakaguchi et al., 2019). As shown in Table 4, the model pre-trained with a sink token performs similarly to that trained using the vanilla approach.

Streaming Performance. As illustrated in Table 3, the streaming perplexities differ between models trained using traditional methods and those augmented with a sink token. Remarkably, the vanilla model requires the addition of multiple tokens as attention sinks to maintain stable streaming perplexity. In contrast, the model trained with a sink token achieves satisfactory streaming performance using just the sink token.

Attention Visualization. Figure 7 contrasts attention maps for models pre-trained with and without a sink token. The model without the sink token, similar to Llama-2-7B (Figure 2), shows early-layer local attention and deeper-layer focus on initial tokens. In contrast, models trained with a sink token consistently concentrate on the sink across layers and heads, indicating an effective attention offloading mechanism. This strong focus on the sink, with reduced attention to other initial tokens, explains the sink token's efficacy in enhancing model's streaming performance.

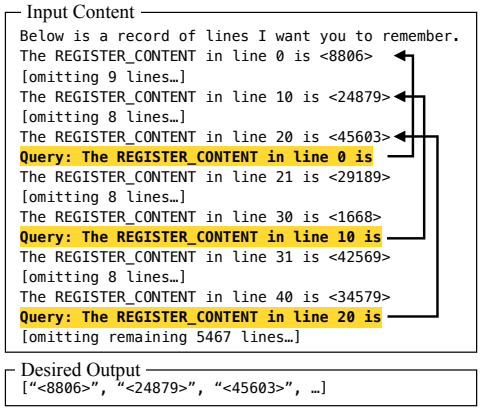

Figure 8: The first sample in StreamEval.

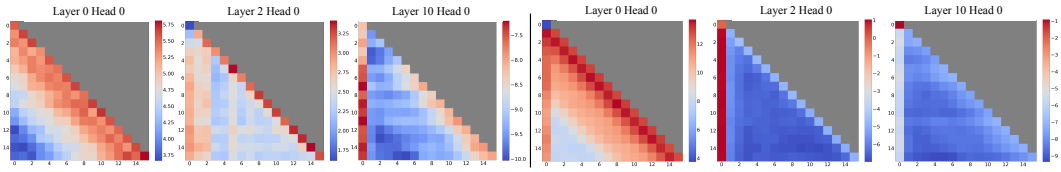

Figure 7: Visualization of average attention logits over 256 sentences, each 16 tokens long, comparing models pre-trained with (left) and without (right) a sink token. Both maps show the same layers and heads. Key observations: (1) Without a sink token, models show local attention in lower layers and increased attention to initial tokens in deeper layers. (2) With a sink token, there is clear attention directed at it across all layers, effectively collecting redundant attention. (3) With the presence of the sink token, less attention is given to other initial tokens, supporting the benefit of designating the sink token to enhance the streaming performance.

Table 5: Accuracy (in %) on the ARC-[Easy, Challenge] datasets. Questions were concatenated and answered in a streaming manner to mimic a real-world chat setting. The dense baseline fails due to Out-of-Memory (OOM) errors. Window attention has poor accuracy. StreamingLLM has comparable results with the one-shot sample-by-sample baseline. Window attention and StreamingLLM use cache sizes of 1024.

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

4.3 Results on Streaming Question Answering with Instruction-tuned Models

To show StreamingLLM's real-world applicability, we emulate multi-round question-answering using instruction-tuned LLMs, commonly used in real-world scenarios.

We first concatenate all question-answer pairs from the ARC-[Challenge, Easy] datasets, feed the continuous stream to Llama-2-[7,13,70]B-Chat models, and assess model completions at each answer position using an exact match criterion. As table 5 indicates, dense attention results in Out-of-Memory (OOM) errors, showing it unsuitable for this setting. While the window attention method works efficiently, it exhibits low accuracy due to random outputs when the input length exceeds the cache size. Conversely, StreamingLLM excels by efficiently handling the streaming format, aligning with the one-shot, sample-by-sample baseline accuracy.

Highlighting a more fitting scenario for StreamingLLM, we introduce a dataset, StreamEval, inspired by the LongEval (Li et al., 2023) benchmark. As depicted in Figure 8, diverging from LongEval's single query over a long-span setup, we query the model every 10 lines of new information. Each query's answer is consistently 20 lines prior, reflecting real-world instances where questions typically pertain to recent information. As illustrated in Figure 9, LLMs employing StreamingLLM maintain reasonable accuracy even as input lengths approach 120K tokens. In contrast, both dense and window attention fail at the pre-training text length and the KV cache size, respectively. Additionally, we utilize two context-extended models, LongChat-7b-v1.5-32k (Li et al., 2023) and Llama-2-7B-32K-Instruct (Together, 2023), to show that StreamingLLM can complement context extension techniques. Within StreamingLLM, context extension means broadening the maximum cache size of streaming LLMs, enabling the capture of broader local information.

4.4 Ablation Studies

Numbers of Initial Tokens. In Table 2, we ablate the effect of adding varying numbers of initial tokens with recent tokens on the streaming perplexity. The results show the insufficiency of introducing merely one or two initial tokens, whereas a threshold of four initial tokens appears enough, with subsequent additions contributing marginal effects. This result justifies our choice of introducing 4 initial tokens as attention sinks in StreamingLLM.

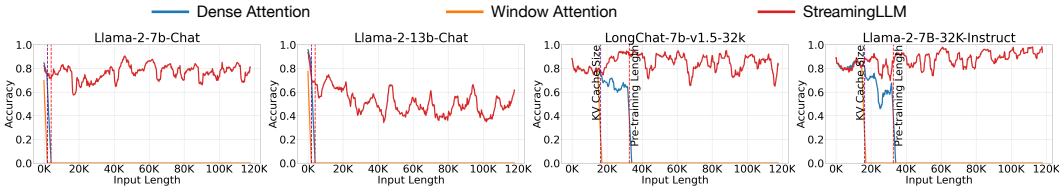

Figure 9: Performance on the StreamEval benchmark. Accuracies are averaged over 100 samples.

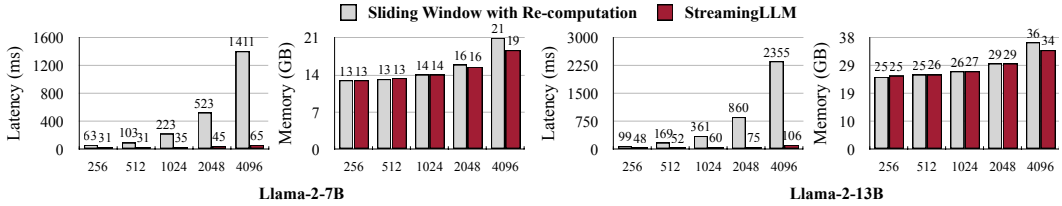

Figure 10: Comparison of per-token decoding latency and memory usage between the sliding window approach with re-computation baseline and StreamingLLM, plotted against the cache size (attention window size) on the X-axis. StreamingLLM delivers a remarkable speedup of up to 22.2× per token and retains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

Cache Sizes. In Table 6, we evaluate cache size's impact on StreamingLLM's perplexity. Contrary to intuition, increasing the cache size doesn't consistently lower the language modeling perplexity. This inconsistency shows a potential limitation where these models might not maximize the utility of the entire context they receive. Future research efforts should target enhancing these models' capabilities to utilize extensive contexts better.

4.5 Efficency Results

We benchmark its decoding latency and memory usage against the sliding window with re-computation, which is the only baseline with acceptable performance. Both methods are implemented using the Huggingface Transformers library (Wolf et al., 2020) and tested on a single NVIDIA A6000 GPU using the Llama-2-7B and Llama-2-13B models. As depicted in Figure 10, as the cache size increases, StreamingLLM's decoding speed demonstrates a linear growth. The sliding window with re-computation baseline has a quadratic rise in decoding latency. Thus, StreamingLLM achieves an impressive speedup, reaching up to 22.2× per token. Despite its reduced latency, StreamingLLM sustains a memory footprint consistent with the re-computation baseline.

Table 6: Effects of cache size on StreamingLLM's performance. Increasing the cache size in StreamingLLM doesn't consistently yield a decrease in perplexity across all models, suggesting these models may not be fully exploiting the provided context. Cache config x+y denotes adding x initial tokens with y recent tokens. Perplexity is evaluated on 400K tokens in the concatenated PG19 test set.

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

5 Conclusion

Deploying LLMs in streaming applications is urgently needed but comes with challenges due to efficiency limitations and reduced performance with longer texts. Window attention provides a partial solution, but its performance plummets when initial tokens are excluded. Recognizing the role of these tokens as "attention sinks", we introduced StreamingLLM —a simple and efficient framework that enables LLMs to handle unlimited texts without fine-tuning. By adding attention sinks with recent tokens, StreamingLLM can efficiently model texts of up to 4 million tokens. We further show that pre-training models with a dedicated sink token can improve the streaming performance. StreamingLLM firstly decouples the LLM's pre-training window size and its actual text generation length, paving the way for the streaming deployment of LLMs.

REFERENCES

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. Falcon-40B: an open large language model with state-of-the-art performance. 2023.

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers, 2023.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. arXiv:2004.05150.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

bloc97. NTK-Aware Scaled RoPE allows LLaMA models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation., 2023. URL https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Quantizable transformers: Removing outliers by helping attention heads do nothing, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023. arXiv: 2306.15595.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness, 2022. arXiv:2205.14135.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Tanya Goyal and Greg Durrett. Evaluating factuality in generation with dependency-level entailment. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online, 2020. Association for Computational Linguistics.

kaiokendev. Things I'm learning while training superhot., 2023. URL https://kaiokendev.github.io/til#extending-context-to-8k.

Ehsan Kamalloo, Nouha Dziri, Charles L. A. Clarke, and Davood Rafiei. Evaluating open-domain question answering in the era of large language models, 2023.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net, April 2020.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, , and Hao Zhang. How long can open-source llms truly promise on context length?, June 2023. URL https://lmsys.org/blog/2023-06-29-longchat.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Evan Miller. Attention is off by one, 2023. URL https://www.evanmiller.org/attention-is-off-by-one.html.

OpenAI. Gpt-4 technical report, 2023.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2023.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *arXiv preprint arXiv:2211.05102*, 2022.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code Llama: Open foundation models for code, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, Jacob Menick, Jiayi Weng, Juan Felipe Ceron Uribe, Liam Fedus, Luke Metz, Michael Pokorny, et al. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2022.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

MosaicML NLP Team. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL www.mosaicml.com/blog/mpt-7b. Accessed: 2023-05-05.

Together. Llama-2-7b-32k-instruct — and fine-tuning for llama-2 models with together api, June 2023. URL https://together.ai/blog/llama-2-7b-32k-instruct.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*. Curran Associates, Inc., 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *CoRR*, abs/1905.07830, 2019. URL http://arxiv.org/abs/1905.07830.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. Benchmarking large language models for news summarization, 2023a.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023b.