Efficient Streaming Language Models with Attention Sinks

Guangxuan Xiao¹, Yuandong Tian², Beidi Chen³, Song Han¹,⁴, Mike Lewis²

Massachusetts Institute of Technology¹
Meta AI²
Carnegie Mellon University³
NVIDIA⁴
Motivation: Use cases
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

w/o StreamingLLM

Model Performance

Breaks

User: Write a C++ program to find the nth Fibonacci number.

Assistant: fibonacci_numbers[n-2] + fibonacci_numbers[n-1]

User: Write a Web API that returns a random joke from a list of 4 jokes.

Assistant: random_joke("joke1", "joke2", "joke3", "joke4")

w/ StreamingLLM

outputs = model.

return forward_call(*args, **kwargs)

File "~/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pacakges/transformers/models/llama/modeling_llama.py", line 820, in forward
outputs = self.model()

return forward_call(*args, **kwargs)

File "~/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pacakges/transformers/models/llama/modeling_llama.py", line 780, in forward
layer_outputs = decoder_layer()

return forward_call(*args, **kwargs)

hidden_states, self_attention_weights, present_key_value = self._attn(

return forward_call(*args, **kwargs)

key_states = torch.cat((past_key_value[0], key_states), dim=2)

torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 50.00 Mib (GPU 0; 47.54 GiB total capacity; 44.53 GiB already allocated; 81.06 Mib free; 46.47 GiB reserved in total by PyTorch) If reserved memory is >= allocated memory try setting max_split_size_mb to avoid fragmentation.

See documentation for Memory Management and PyTorch_CUDA_ALLOC_CONF (streaming) guangxuan@u92:/workspace/streaming-llms
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 Keys and Values 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

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

\[
BS \times 32 \times 32 \times 128 \times N \times 2 \times 2\text{bytes} = 0.5\text{MB} \times BS \times N
\]

• Now we calculate the KV cache size under \( BS = 4 \) and different sequence lengths.
  • Quickly larger than model weights

![Graph showing KV cache size (GB) vs. Sequence Length](chart.png)

\( 4 \times 32 \times 32 \times 128 \times 32K \times 2 \times 2 = 64\text{GB} \)
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

\[ O(TL) \]  \[ \text{PPL: 5158} \]

Breaks when initial tokens are evicted.

Llama-2-7B

Input Length

log PPL
**Difficulties of Other Methods**

(a) Dense Attention

- $O(T)$ ✗
- PPL: 5641 ✗

KV cache size grows linearly with the sequence length; **perplexity explodes** after exceeding the max context length.

(b) Window Attention

- $O(1)$ ✓
- PPL: 5158 ✗

KV cache size is constant; but **perplexity explodes** after sequence length exceeds the KV cache size (first token evicted).

(c) StreamingLLM (ours)

- $O(1)$ ✓
- PPL: 5.40 ✓

perplexity doesn’t explode; KV cache size is **constant**.
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.

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

• 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>
<thead>
<tr>
<th>Llama-2-13B</th>
<th>PPL (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 + 1024 (Window)</td>
<td>5158.07</td>
</tr>
<tr>
<td>4 + 1020</td>
<td>5.40</td>
</tr>
<tr>
<td>4&quot;\n&quot;+1020</td>
<td>5.60</td>
</tr>
</tbody>
</table>

\[P(S) = P(\text{Where}) \times P(\text{are} | \text{Where}) \times P(\text{we} | \text{Where are}) \times P(\text{going} | \text{Where are we})\]
• **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)**

\[O(TL) \checkmark \quad \text{PPL: 5.40} \checkmark\]

Can perform efficient and stable language modeling on long texts.
• Use positions *in the cache* instead of those *in the original text*. 

![Diagram showing positional encoding assignment](image)

- Generating Token 9
- Attention Sinks
- Evicted Tokens
- Rolling KV Cache

- Assigned Positions
- 4 5 6 7
Streaming Performance

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

- Dense attention fails beyond pre-training attention window size.
- Window attention fails after input exceeds cache size (initial tokens evicted).
- StreamingLLM shows stable performance; perplexity close to 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.
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>
<thead>
<tr>
<th>Cache Config</th>
<th>0+2048</th>
<th>1+2047</th>
<th>2+2046</th>
<th>4+2044</th>
<th>8+2040</th>
</tr>
</thead>
<tbody>
<tr>
<td>Falcon-7B</td>
<td>17.90</td>
<td>12.12</td>
<td>12.12</td>
<td>12.12</td>
<td>12.12</td>
</tr>
<tr>
<td>MPT-7B</td>
<td>460.29</td>
<td>14.99</td>
<td>15.00</td>
<td>14.99</td>
<td>14.98</td>
</tr>
<tr>
<td>Pythia-12B</td>
<td>21.62</td>
<td>11.95</td>
<td>12.09</td>
<td>12.09</td>
<td>12.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache Config</th>
<th>0+4096</th>
<th>1+4095</th>
<th>2+4094</th>
<th>4+4092</th>
<th>8+4088</th>
</tr>
</thead>
<tbody>
<tr>
<td>Llama-2-7B</td>
<td>3359.95</td>
<td>11.88</td>
<td>10.51</td>
<td>9.59</td>
<td>9.54</td>
</tr>
</tbody>
</table>
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.

![Graph showing training loss](image)

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

<table>
<thead>
<tr>
<th>Cache Config</th>
<th>0+1024</th>
<th>1+1023</th>
<th>2+1022</th>
<th>4+1020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>27.87</td>
<td>18.49</td>
<td>18.05</td>
<td>18.05</td>
</tr>
<tr>
<td>Zero Sink</td>
<td>29214</td>
<td>19.90</td>
<td>18.27</td>
<td>18.01</td>
</tr>
<tr>
<td>Learnable Sink</td>
<td>1235</td>
<td><strong>18.01</strong></td>
<td>18.01</td>
<td>18.02</td>
</tr>
</tbody>
</table>
Attention Sinks in Other Transformers

Encoder Models: ViT and BERT

Vision Transformers Need Registers
Can StreamingLLM give us infinite context?

- Non-stop chatting ≠ Infinite context
- Tokens that are evicted from cache cannot be attended.

<table>
<thead>
<tr>
<th>Line Distances</th>
<th>Token Distances</th>
<th>Cache Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>460</td>
<td>85.80 84.60 81.15 77.65</td>
</tr>
<tr>
<td>40</td>
<td>920</td>
<td>80.35 83.80 81.25 77.50</td>
</tr>
<tr>
<td>60</td>
<td>1380</td>
<td>79.15 82.80 81.50 78.50</td>
</tr>
<tr>
<td>80</td>
<td>1840</td>
<td>75.30 77.15 76.40 73.80</td>
</tr>
<tr>
<td>100</td>
<td>2300</td>
<td>0.00 61.60 50.10 40.50</td>
</tr>
<tr>
<td>150</td>
<td>3450</td>
<td>0.00 68.20 58.30 38.45</td>
</tr>
<tr>
<td>200</td>
<td>4600</td>
<td>0.00 0.00 62.75 46.90</td>
</tr>
<tr>
<td>400</td>
<td>9200</td>
<td>0.00 0.00 0.00 45.70</td>
</tr>
<tr>
<td>600</td>
<td>13800</td>
<td>0.00 0.00 0.00 28.50</td>
</tr>
<tr>
<td>800</td>
<td>18400</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>1000</td>
<td>23000</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
</tbody>
</table>
Thanks for Listening!

- We propose StreamingLLM, enabling the streaming deployment of LLMs.
- Demo: https://youtu.be/UgDcZ3rvRPG