

Efficient Large Language Models and Generative Al

Guangxuan Xiao

xgx@mit.edu https://guangxuanx.com

Large Language Models (LLMs) are Powerful



Transformer-based

Software Development



Scientific Discovery



Disability Aid





LLM Models Outgrow Hardware Capacity **Reducing LLM serving cost is essential**

- LLM sizes and computation are increasing exponentially.
- bridge the gap.



Domain-specific accelerator alone is not enough; we need efficient deployment algorithms to



Content

Today I will introduce:

- LLM Compression & Inference Speedup
 - Language Models (ICML 2023)
- Efficient Application of LLMs for Extended Inputs
 - (ICLR 2024)

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large

Efficient Streaming Language Models with Attention Sinks (StreamingLLM)









SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models

Guangxuan Xiao^{*1}, Ji Lin^{*1}, Mickael Seznec², Hao Wu², Julien Demouth², Song Han¹

Massachusetts Institute of Technology¹ NVIDIA²

Quantization Can Reduce Deployment Costs

Quantization lowers the bit-width and improves efficiency

- Serving a 175B GPT-3 model at least requires:
 - FP16: 350GB memory 5 x 80GB A100 GPUs
 - INT8: 175GB memory **3 x 80GB A100 GPUs**









Naive Quantization Methods are Inaccurate Activation outliers destroy quantized performance



- W8A8 quantization has been an industrial standard for CNNs, but not LLM. Why?
- Systematic outliers emerge in activations when we scale up LLMs beyond 6.7B. Traditional CNN quantization methods will destroy the accuracy.

175B

Model Size



Understanding the Quantization Difficulty of LLMs Smoothing activation to reduce quantization error



Weights are easy to quantize, but activation is hard due to outliers



8

Understanding the Quantization Difficulty of LLMs **Smoothing activation to reduce quantization error**



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels



9

Understanding the Quantization Difficulty of LLMs **Smoothing activation to reduce quantization error**



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels
- Migrate the quantization difficulty from activation to weights, so both are easy to quantize





System Implementation **Efficient System Implementation**

- We integrate SmoothQuant into FasterTransformer
- All compute-intensive operators (Linear, BMM) are quantized









SmoothQuant is Accurate and Efficient

- SmoothQuant well maintains the accuracy without fine-tuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.

LAMBADA Accuracy

	OPT-175B	BLOOM-176B	GLM-130B
FP16	71.6%	68.2%	73.8%
SmoothQuant	71.2%	68.3%	73.7%



12

Industry & Community Impact SmoothQuant is widely adopted by industry

- NVIDIA FasterTransformer
- NVIDIA TRT-LLM
- MLPerf 8-bit: closed the accuracy gap.
- Intel Neural Compressor / Q8-Chat on Xeon
- Meta/Microsoft/Amazon/HuggingFace …









Efficient Streaming Language Models with Attention Sinks

Guangxuan Xiao¹, Yuandong Tian², Beidi Chen³, Song Han^{1,4}, Mike Lewis²

Massachusetts Institute of Technology¹ Meta Al² Carnegie Mellon University³ NVIDIA⁴

Challenges of Deploying LLMs in Streaming Applications

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



"Alexa, open Showcase Cinemas."

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



https://github.com/tomaarsen/attention_sinks





Challenges of Deploying LLMs in Streaming Applications w/ StreamingLLM w/o StreamingLLM

S=0 python examples/run_streaming_llama.py Loading model from lmsys/vicuna-13b-v1.3 ... Loading checkpoint shards: 67%

- (streaming) guangxuan@l29:~/workspace/streaming-llm\$ CUDA_VISIBLE_DEVICE (streaming) guangxuan@l29:~/workspace/streaming-llm\$ CUDA_VISIBLE_DEVICES=1 py thon examples/run_streaming_llama.py --enable_streaming
 - Loading model from lmsys/vicuna-13b-v1.3 ...
 - 2/3 [00:09<00:04, 4.94s/it] Loading checkpoint shards: 67%]

| 2/3 [00:09<00:04, 4.89s/it]</pre>





16

Challenges of Deploying LLMs in Streaming Applications

w/o StreamingLLM

	"0000"	"0"0"000	0000"						
	00000000	aaaa"a"a	"0"0000"	a"a"a"""		aaaaaaaa	aaaaaaa		
0000000 a''''	000000000	000000000	0000"000	00000000	000"0"""			·····0-1	t0
0									

Model Performance ASSISTANT: 0000000-t-t-t-t"

USER: Write a C++ prog Break Sibonacci number using recursi on.

USER: Now we define a sequence of numbers in which each number is the su m of the three preceding ones. The first three numbers are 0, -1, -1. Wr ite a program to find the nth number.

ASSISTANT: 0-a-a-eah00000000000

USER: Write a simple website in HTML. When a user clicks the button, it shows a random joke from a list of 4 jokes.

w/o StreamingLLM outputs = model(File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/torch/nn/modules/module.py", line 1501, in _call_impl return forward_call(*args, **kwargs) File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/transformers/models/llama/modeling_llama.py", line 820, in forward outputs = self.model(File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/torch/nn/modules/module.py", line 1501, in _call_impl return forward_call(*args, **kwargs) File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/transformers/models/llama/modeling_llama.py", line 708, in forward layer_outputs = decoder_layer(File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/tor _______ es/mod _____ retu 🚺 f File "/ Jython3.8/site-pac kages/transformers/models/llama/modeling_llama.py", ____ne 424, in forward hidden_states, self_attn_weights, present_key_value = self.self_attn File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/torch/nn/modules/module.py", line 1501, in _call_impl return forward_call(*args, **kwargs) File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-pac kages/transformers/models/llama/modeling_llama.py", line 337, in forward key_states = torch.cat([past_key_value[0], key_states], dim=2) torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 90.00 MiB (GPU 0; 47.54 GiB total capacity; 44.53 GiB already allocated; 81.0 6 MiB free; 46.47 GiB reserved in total by PyTorch) If reserved memory i s >> allocated memory try setting max_split_size_mb to avoid fragmentati on. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF (streaming) guangxuan@l29:~/workspace/streaming-llm\$





The Limits of Window Attention

- A natural approach window attention: caching only the most recent Key-Value states. lacksquare
- is evicted. (b) Window Attention



Drawback: model collapses when the text length surpasses the cache size, when the initial token







The "Attention Sink" Phenomenon



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.

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

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.







StreamingLLM: Using Attention Sinks for Infinite Streams

- without additional training.
- stabilize the model's behavior.



(d) StreamingLLM (ours)

Can perform efficient and stable language modeling on long texts.

Objective: Enable LLMs trained with a finite attention window to handle infinite text lengths

Key Idea: preserve the KV of attention sink tokens, along with the sliding window's KV to









Streaming Performance



- Dense attention fails beyond pre-training attention window size.
- Window attention fails after input exceeds cache size (initial tokens evicted).
- \bullet



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

StreamingLLM shows stable performance; perplexity close to sliding window with re-computation baseline.





Streaming Performance Super Long Language Modeling

lacksquaremodel up to 4 million tokens.





With StreamingLLM, model families include Llama-2, MPT, Falcon, and Pythia can now effectively







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.



Llama-2-7B

Llama-2-13B







Conclusion

- We propose StreamingLLM, enabling the streaming deployment of LLMs. \bullet
- Paper: <u>https://arxiv.org/abs/2309.17453</u>
- Code: <u>https://github.com/mit-han-lab/streaming-llm</u> 6.7K Stars ${\bullet}$
- Demo: <u>https://youtu.be/UgDcZ3rvRPg</u>
- Integration: ${\bullet}$
 - NVIDIA TensorRT-LLM
 - Intel Transformer Extension lacksquare
 - Huggingface Transformers





Thanks for Listening!



We need efficient algorithms to bridge the gap.

