H2O: Heavy Hitter Oracle for Efficient Generative Inference of Large Language Models

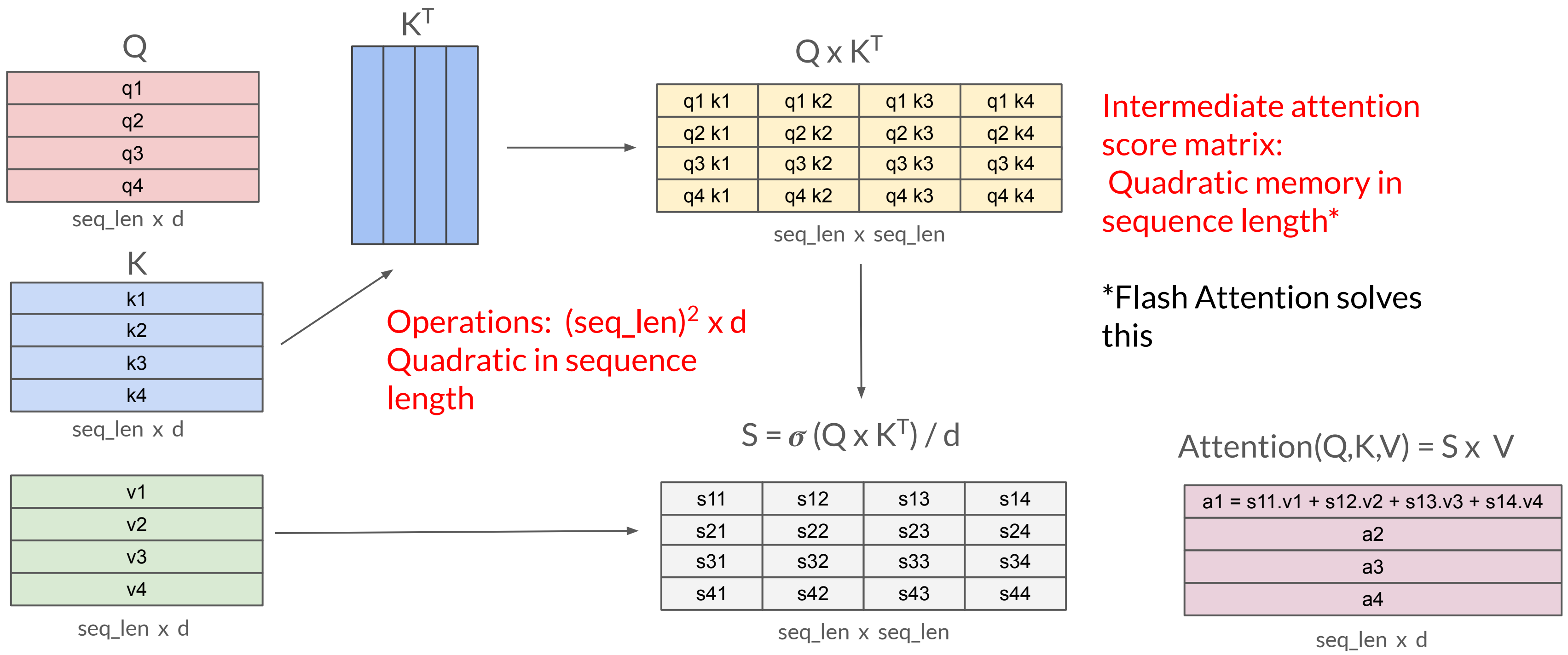
Zhang et al., (NeurIPS 2023)

- Neel Dani
(neeld2@illinois.edu)

Agenda

- **<u>Attention mechanism</u>**
- Related work: Sparse Attention
- H2O - Motivation
- H2O - Challenges
- H2O - Algorithm
- Experiments
- Thoughts

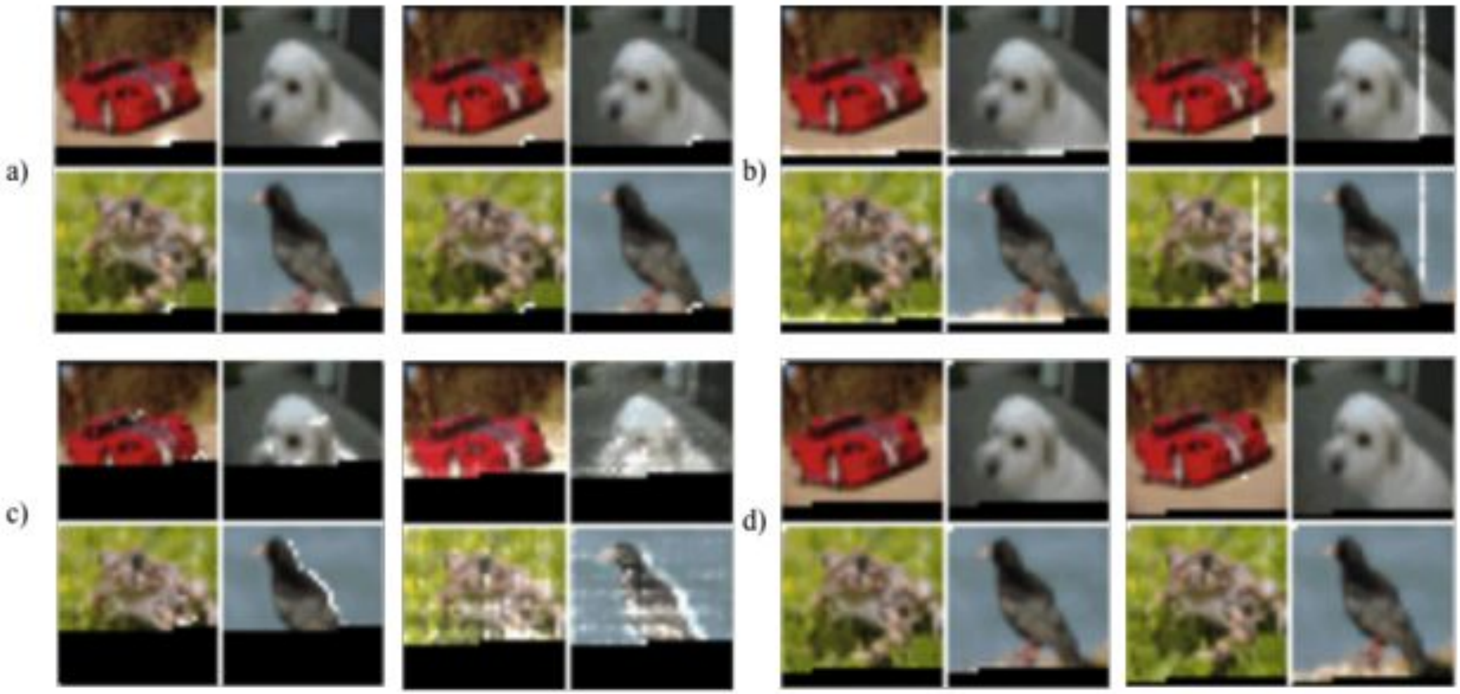
LLM Inference (Prefill phase)



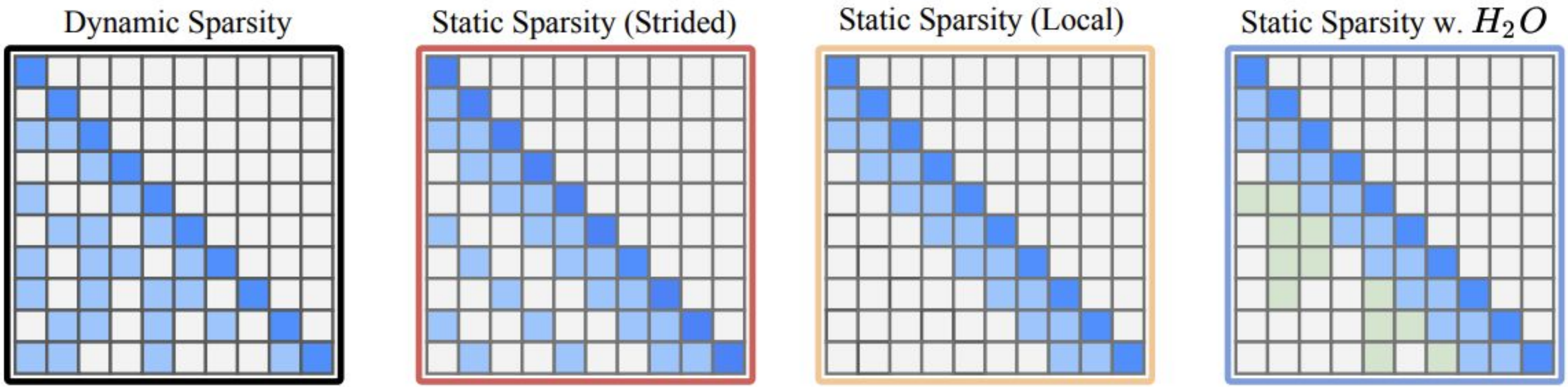
Agenda

- Attention mechanism
- **Related work: Sparse Attention**
- H2O - Motivation
- H2O - Challenges
- H2O - Algorithm
- Experiments
- Thoughts

(Related Work) Sparse Attention: Intuition



(Related Work) Sparse Attention: Factorization

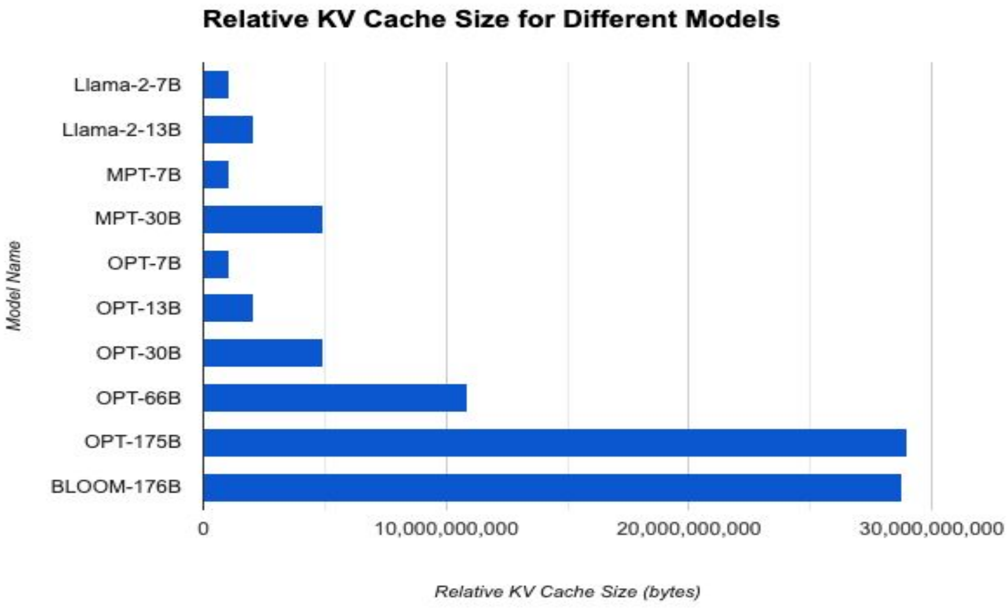


* There have been a lot of new connectivity algorithms since: *Local attention*: O(seq_len * W), *Random attention*: O(seq_len * R), *Sparse (Strided + Fixed)*: O(seq_len * √seq_len), *Block*, *Global* etc

Agenda

- Attention mechanism
- Related work: Sparse Attention
- **H2O - Motivation**
- H2O - Challenges
- H2O - Algorithm
- Experiments
- Thoughts

Motivation: KV Cache can grow significantly



KV cache size: (2 x B x L x S x d x P) bytes

where:
B = batch size
L = # of layers
S = sequence length
D = hidden dimension
P = precision

LLaMA-2-7B with a batch size of 64 and sequence length of 1024 requires a cache size of 32GB

NVIDIA A100 GPU has ~ 40 GB - 80 GB

Agenda

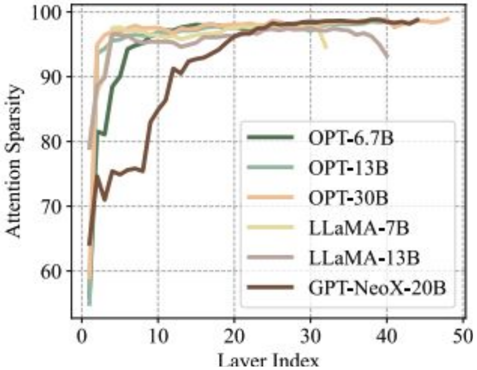
- Attention mechanism
- Related work: Sparse ATtention
- H2O - Motivation
- <u>H2O - Challenges</u>
- H2O - Algorithm
- Experiments
- Thoughts

Problems with cache eviction

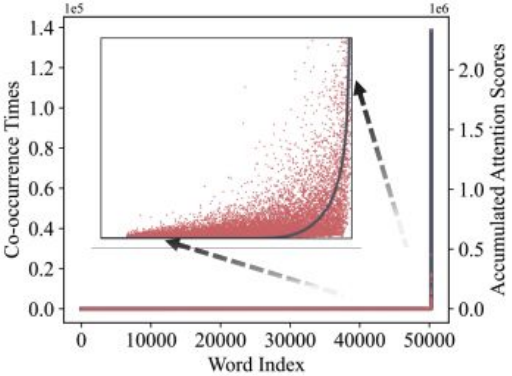
- **Small cache size**
  - Challenge: Each decoding step requires all previous KV tensors
- **Low cache miss rate: Maintain accuracy**
  - Challenge: Combinatorial problem
- **Low-cost eviction policy:**
  - Challenge: Brute forced policy might have a high latency

Observations for cache eviction

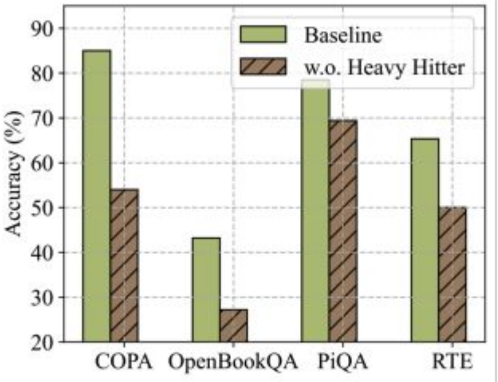
#1: Attention scores are 95% sparse



#2: Distribution of the attention scores for a word follows the power-law



#3: A small portion of the tokens contribute most to the attention score



Problem formulation: We maintain a cache of constant size *k* by evicting at most 1 KV from the cache. Further, the evicted KV is set to 0 and subtracted when computing the normalization.

We do not know heavy-hitters beforehand

1. But we do not have the attention scores of the entire sequence beforehand (LLM generation is autoregressive)

2. Formulate KV Cache eviction as a dynamic submodular problem and use a greedy policy for cache eviction

Diminishing return property:

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y), \textit{ where } f(\cdot) := F(Z, \cdot). \quad X \subset Y,$$

Theoretical guarantee (under mild assumption):

$$f(\widetilde{S}_i) \geq (1 - \alpha)(1 - 1/e) \max_{|S|=k} f(S) - \beta,$$

Agenda

- Attention mechanism
- Related work: Sparse ATtention
- H2O - Motivation
- H2O - Challenges
- **H2O - Algorithm**
- Experiments
- Thoughts

Algorithm

Algorithm 1 H_2 Eviction Algorithm

procedure $\text{H}_2\text{\_EVICTION}(Q, K \in \mathbb{R}^{n\times d}, k \in \mathbb{N})$
  Let k denote the budget size of cache
  $S_0 \leftarrow \emptyset$
  for $i = 1 \to n$ **do**
    if $i \leq k$ **then**
      $S_i \leftarrow S_{i-1} \cup \{i\}$
    else
      $D_i \leftarrow (\exp(Q_{i,*}(K_{S_{i-1},*})^\top) - 1_{[i]\setminus S_{i-1}}) \cdot \mathbf{1}_i$
      $o_i \leftarrow D_i^{-1} \cdot (\exp(Q_{i,*}(K_{S_{i-1},*})^\top) - 1_{[i]\setminus S_{i-1}})$
      $F_{\text{score}}(T) := \sum_{s\in T} o_s$
      $G_i \leftarrow S_{i-1} \cup \{i\}$
      $u \leftarrow \arg\max_{v\in G_i} F_{\text{score}}(S_{i-1} \cup \{i\}\setminus\{v\})$
      $S_i \leftarrow (S_{i-1} \cup \{i\})\setminus\{u\}$
    end if
  end for
end procedure

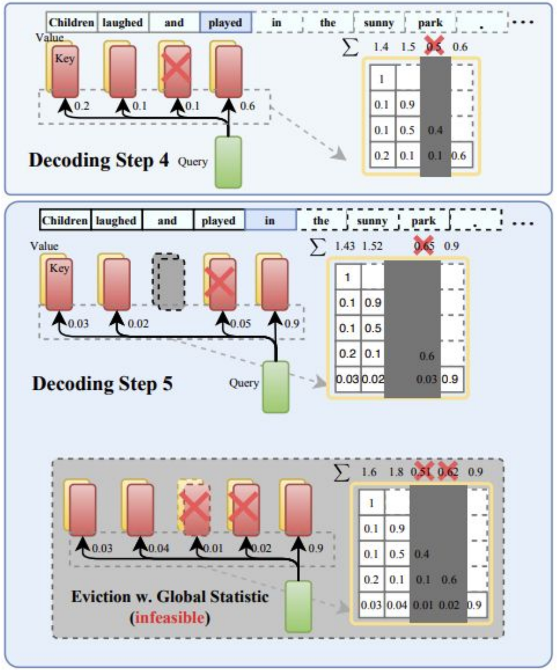
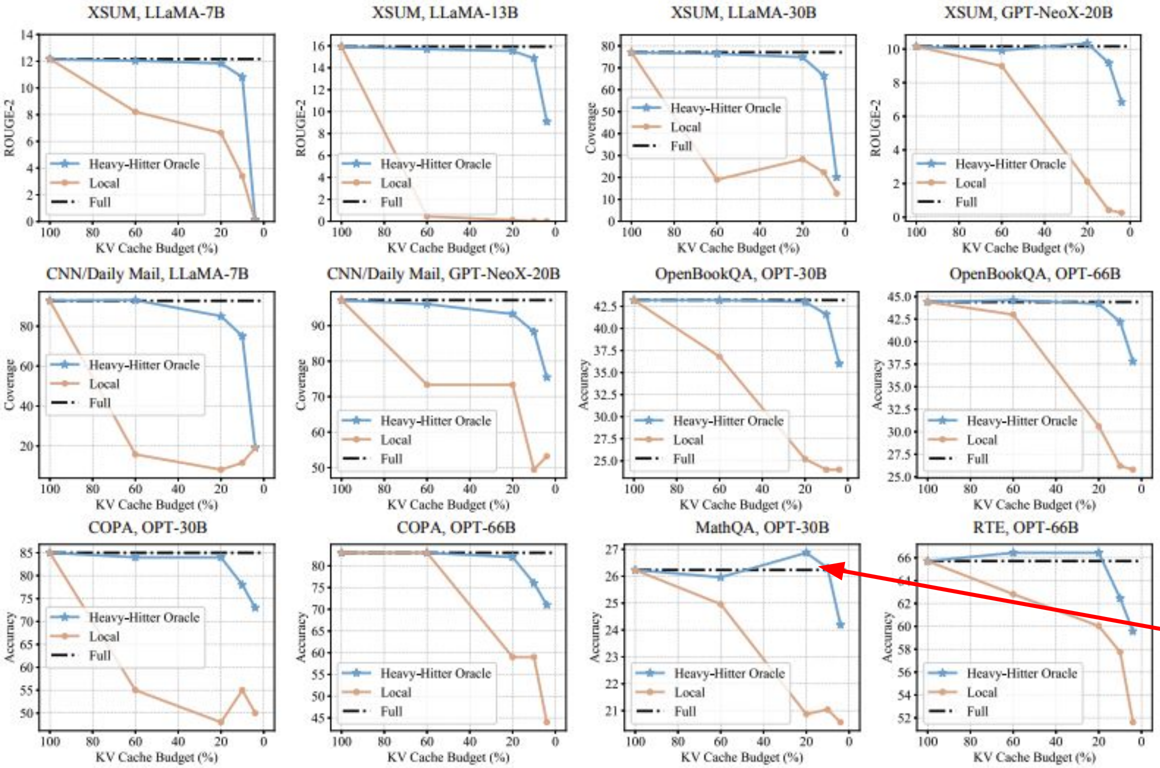


Figure 3: Illustration of Algorithm 1 during two consecutive decoding steps.

Agenda

- Attention mechanism
- Related work: Sparse ATtention
- H2O - Motivation
- H2O - Challenges
- H2O - Algorithm
- <u>**Experiments**</u>
- Thoughts

Experiments



1. Experiments on OPT, LLaMA, GPT-NeoX-20B

2. 8 tasks sampled from HELM and lm-eval-harness (COPA, MathQA, OpenBook, PiQA, RTE, XSUM, CNN and Winogrande

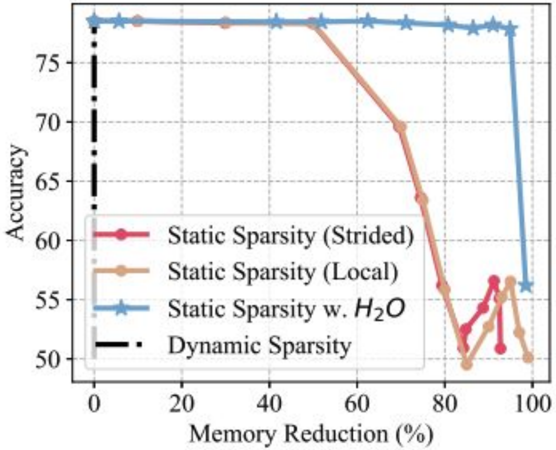
3. Baselines: Sparse Transformer (strided and fixed), “Local” strategy, full KV cache with fewer shots

Performs better than baselines: Authors claim this is due to a regularization effect

Experiments (Contd)

Table 2: Results of different sparsification methods *w.* or *w.o.* H_2. Experiments are conducted with OPT-30B with 20% KV cache budget.

Models	COPA	OpenBookQA	PiQA	Winogrande
Full	85.00	43.20	78.51	70.24
Local *w.o.* H_2	48.00	25.20	55.82	49.17
Local *w.* H_2	84.00	43.00	78.45	69.06
Sparse Transformer (strided) *w.o.* H_2	50.00	24.60	56.20	47.59
Sparse Transformer (strided) *w.* H_2	83.00	42.60	78.24	69.61
Sparse Transformer (fixed) *w.o.* H_2	61.00	23.80	58.60	49.88
Sparse Transformer (fixed) *w.* H_2	76.00	41.40	77.80	64.96



Combining heavy-hitters retention with other strategies improves accuracy

System Performance

Table 3: Generation throughput (token/s) on a T4 GPU with different systems. In the sequence length row, we use "512 + 32" to denote a prompt length of 512 and a generation length of 32. "OOM" means out-of-memory. The gray text in the bracket denotes the effective batch size and the lowest level of the memory hierarchy that the system needs for offloading, where "C" means CPU and "G" means GPU.

Seq. length	512+32		512+512		512+1024	
Model size	6.7B	30B	6.7B	30B	6.7B	30B
Accelerate	20.4 (2, G)	0.6 (8, C)	15.5 (1, G)	0.6 (8, C)	5.6 (16, C)	0.6 (8, C)
DeepSpeed	10.2 (16, C)	0.6 (4, C)	9.6 (16, C)	0.6 (4, C)	10.1 (16, C)	0.6 (4, C)
FlexGen	20.2 (2, G)	8.1 (144, C)	16.8 (1, G)	8.5 (80, C)	16.9 (1, G)	7.1 (48, C)
H_2O (20%)	35.1 (4, G)	12.7 (728, C)	51.7 (4, G)	18.83 (416, C)	52.1 (4, G)	13.82 (264, C)

Table 5: Generation throughput and latency on an A100 GPU. In the sequence length row, we use "7000 + 1024" to denote a prompt length of 7000 and a generation length of 1024. "OOM" means out-of-memory.

Seq. length	Model size	Batch size	Metric	FlexGen	H_2O (20%)
7000+1024	30B	1	latency (s)	57.0	50.4
5000+5000	13B	4	latency (s)	214.2	155.4
2048+2048	6.7B	24	latency (s)	99.5	53.5
2048+2048	6.7B	24	throughput (token/s)	494.1	918.9
2048+2048	6.7B	64	throughput (token/s)	OOM	1161.0

1. Experiments on OPT 6.7B and OPT 30B
2. Use CPU offloading if model does not fit on the NVIDIA T4 (16GB) and NVIDIA A100 (80GB)
3. DeepSpeed ZeRO-Inference, Hugging Face Accelerate and FlexGen used as baselines

Agenda

- Attention mechanism
- Related work: Sparse ATtention
- H2O - Motivation
- H2O - Challenges
- H2O - Algorithm
- Experiments
- Thoughts

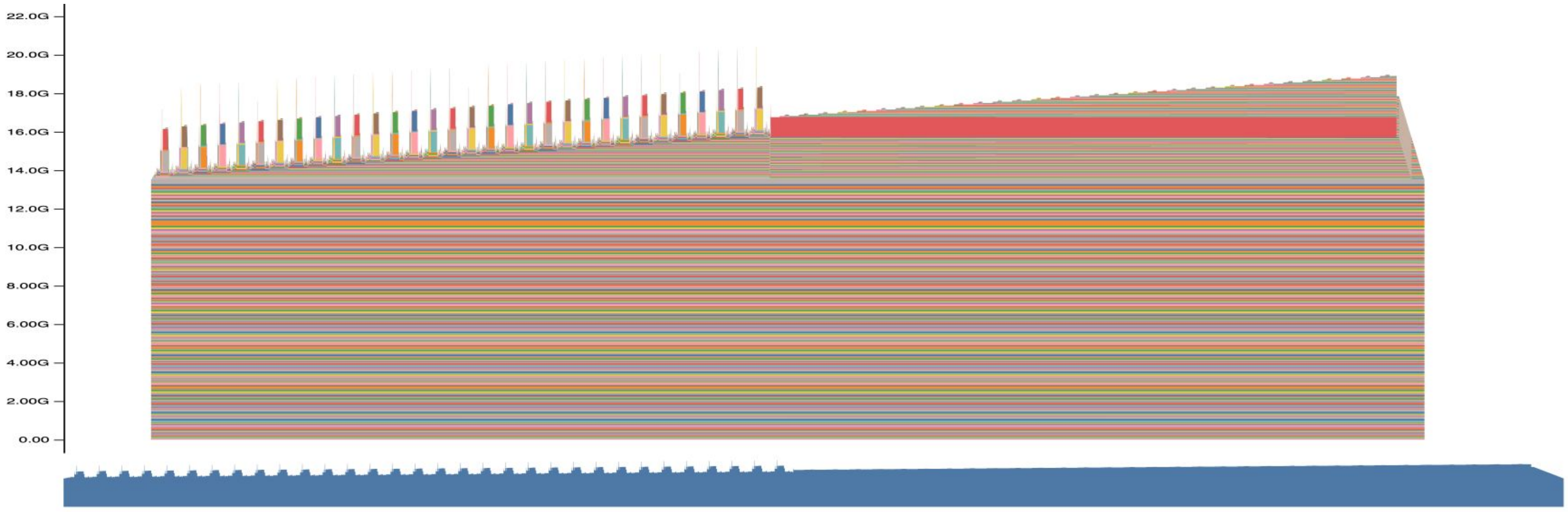
Thoughts

- **Strengths**
  - Great results with 20% cache budget
  - Extensive evaluation which also provides system level benefits
  - The design allows generation of infinite tokens

- **Weaknesses**
  - Can’t take advantage of Flash Attention leading to materialization of Attention score matrix which scales quadratically with sequence length

- **Future Directions**
  - Quantization of KV Cache with H2O
  - IO-aware kernel (similar to Flash Attention) which also computes the accumulated attention scores
  - Heavy-hitters in the MLP layer

Thoughts (Contd)



Active memory usage for prefill and decoding on Llama-2-7B chat for batch size = 1, prompt length = 2048, output length = 1024

References

1. Generating Long Sequences with Sparse Transformers (link)
2. Sparsity in Transformers (link)
3. Flash Attention (link)
4. Dynamic Submodular Maximization (link)