Sparsity for Efficient Long Sequence Generation

Beidi Chen (CMU / FAIR)

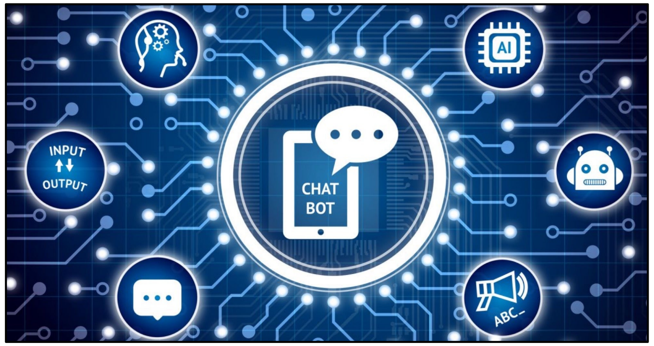
H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *NeurIPS 2023*. Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, Beidi Chen.
https://github.com/FMInference/H2O

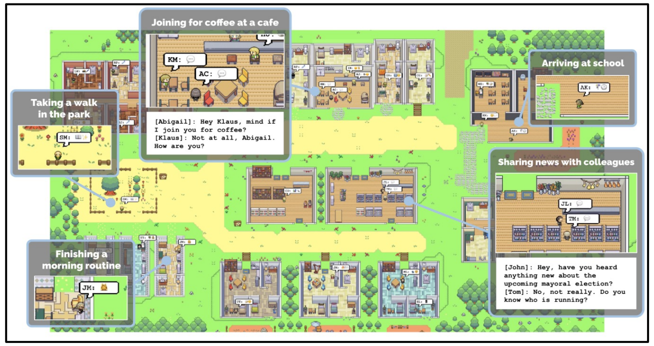
StreamingLLM: Efficient Streaming Language Models with Attention Sinks. Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, Mike Lewis.
https://github.com/mit-han-lab/streaming-llm

Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. *ICML 2023 (Oral).* Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, Beidi Chen.
https://github.com/FMInference/DejaVu

Compress, Then Prompt: Improving Accuracy-Efficiency Trade-off of LLM Inference with Transferable Prompt. Zhaozhuo Xu, Zirui Liu, Beidi Chen, Yuxin Tang, Jue Wang, Kaixiong Zhou, Xia Hu, Anshumali Shrivastava.

LLMs are Powerful

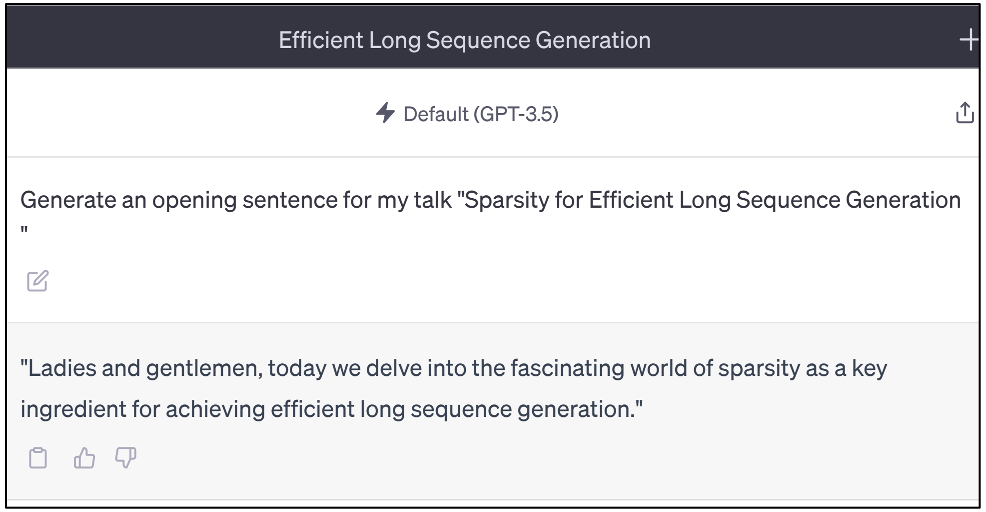


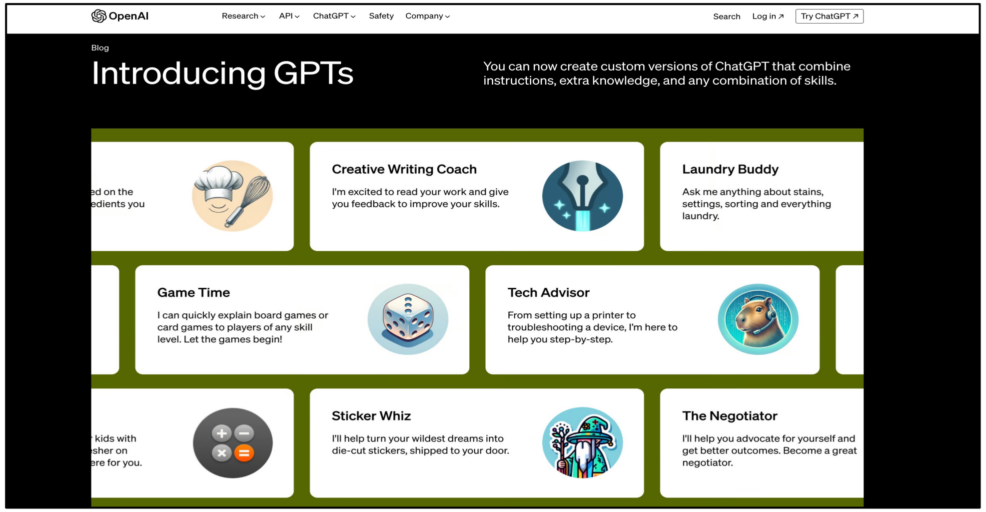


Conversational AI

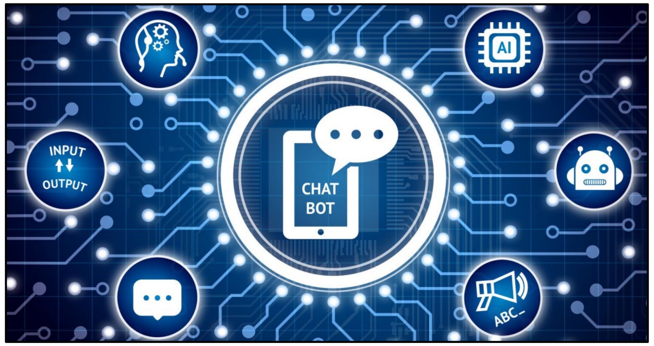
Content Generation

AI Agents



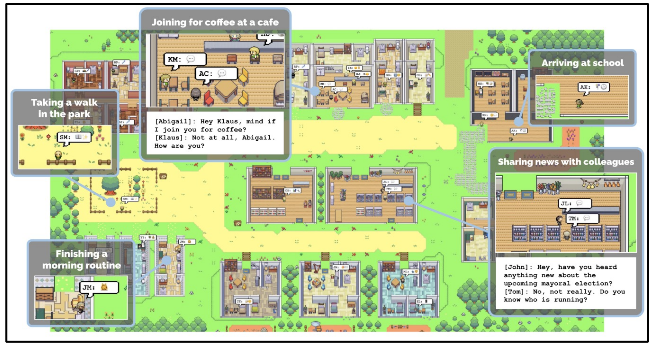


LLMs are Powerful , but Very Expensive to Deploy



Conversational AI

Content Generation

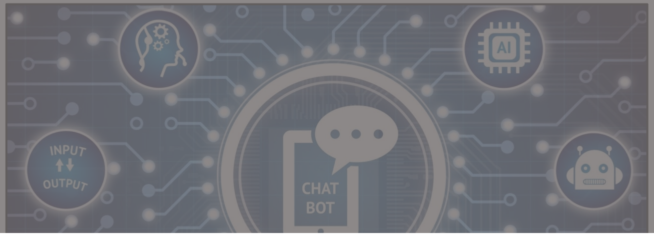


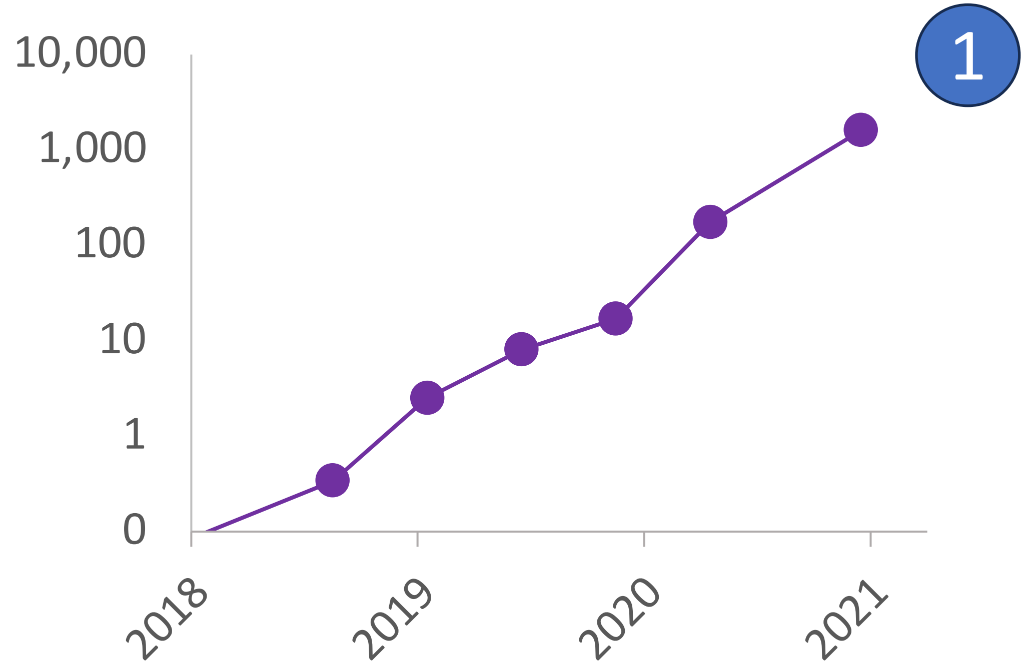
AI Agents

Major Challenges: memory IO (*Pope et al.*) + limited context window

- large mem, e.g. a Llama2-70B model needs
  - 140 GB for parameters,
  - 160 GB for activation (KV cache),
    even with Multi-Group-Attention (8K seqlen + 64 batch size)
- low parallelizability, e.g. generate 100 tokens -> load model, KV cache 100 times
- Perplexity explosion beyond pre-trained windows

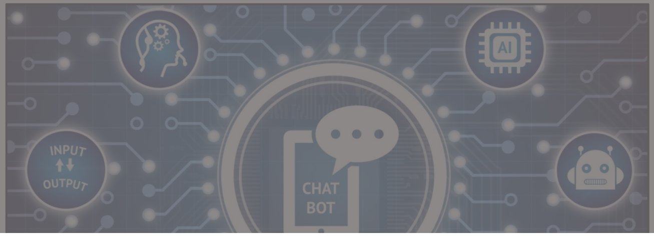
LLMs are Powerful , but Very Expensive to Deploy

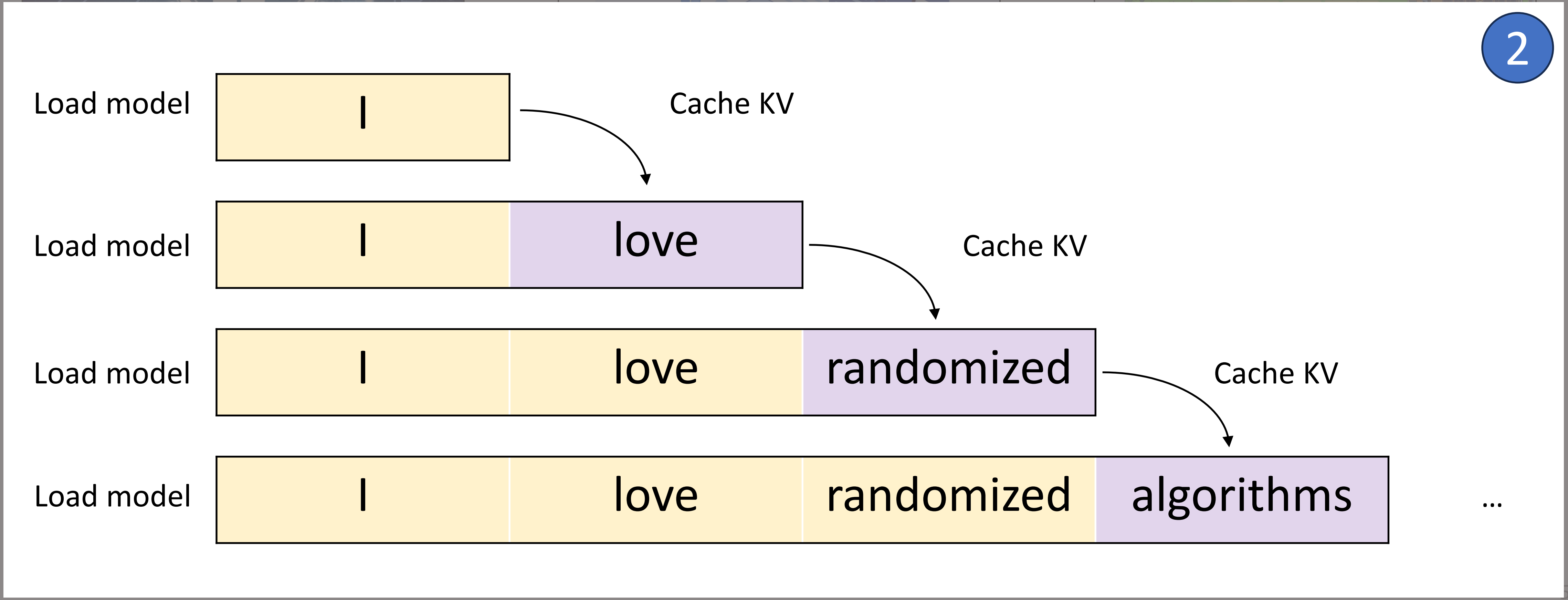




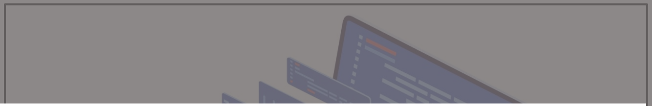
Exponential model size

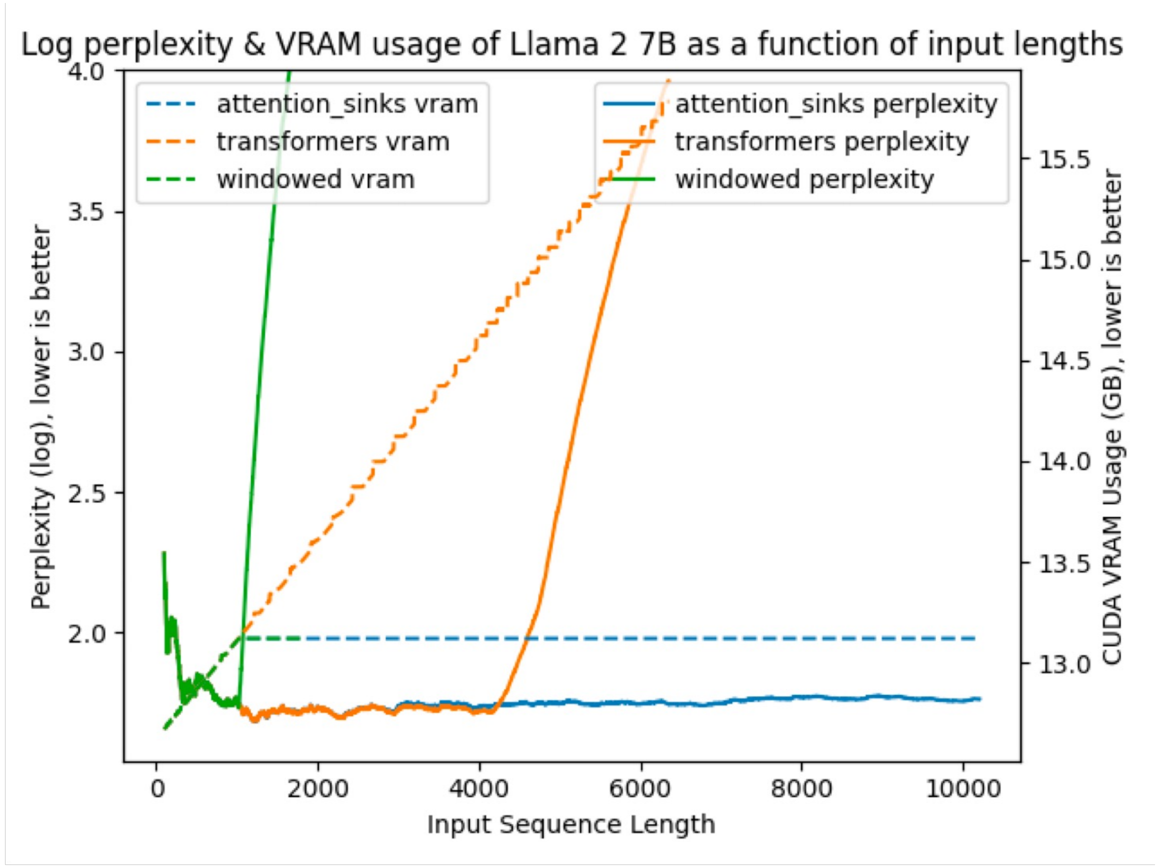
LLMs are Powerful , but Very Expensive to Deploy





LLMs are Powerful , but Very Expensive to Deploy



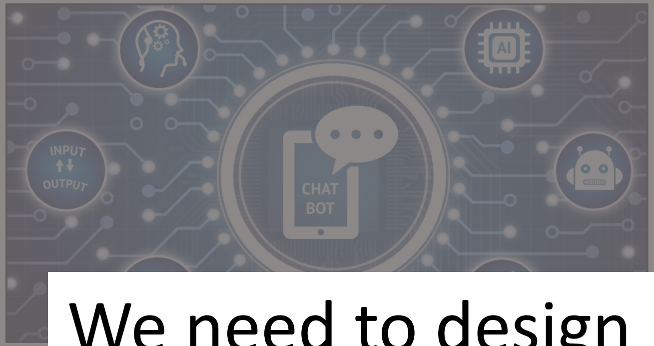


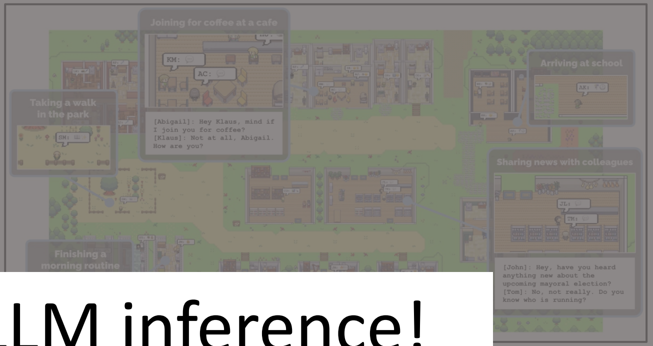
AI Agents

en + 64 batch size)

oad model, KV cache 100 times

LLMs are Powerful , but Very Expensive to Deploy





We need to design more efficient algorithms for LLM inference!

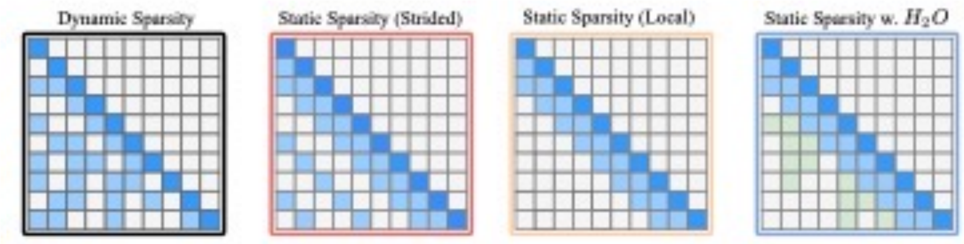
AI Agents

Major bottleneck: memory IO (*Pope et al.*)

- large mem, e.g. a Llama2-70B model needs
  - 140 GB for weights,
  - 160 GB for KV cache even with MGA (8K seqlen + 64 batch size)
- low parallelizability, e.g. generate 100 tokens -> load model, KV cache 100 times

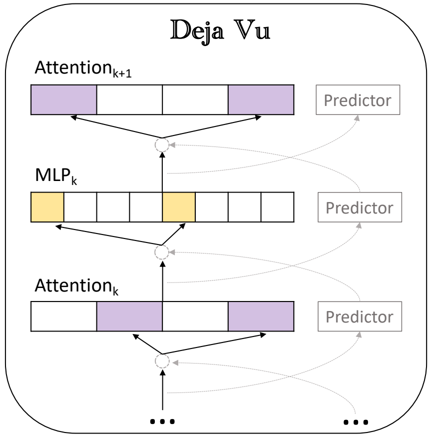
LLMs are Powerful , but Very Expensive to Deploy

H_2O (*NeurIPS'23*)



29x, 29x, 3x higher throughput, 1.9x lower latency than DeepSpeed Zero-Inference, HuggingFace Accelerate, and FlexGen with Heavy-Hitter Sparsity.

Deja Vu (*ICML'23*)



2x lower latency than FasterTransformer and 6x than HuggingFace on 8xA100 with contextual sparsity.

StreamingLLM (*new* 🔥)

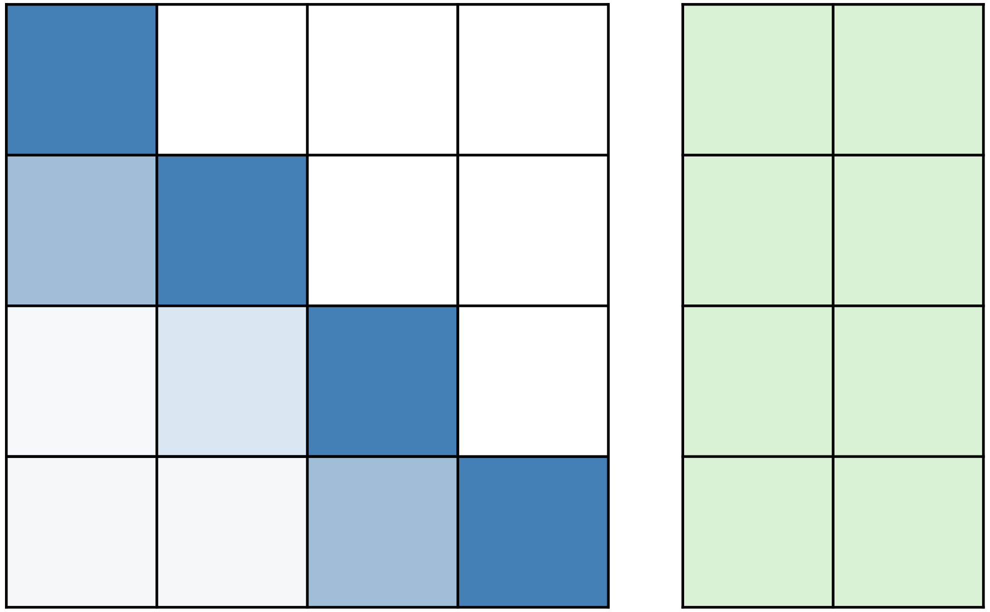
Model 4 million tokens… 22x faster than sliding window recomputation with Attention Sink.

Compress, Then Prompt (*new* 🔥)

8x extreme model compression (Sparse+Quantize) with Prompt Recovery.

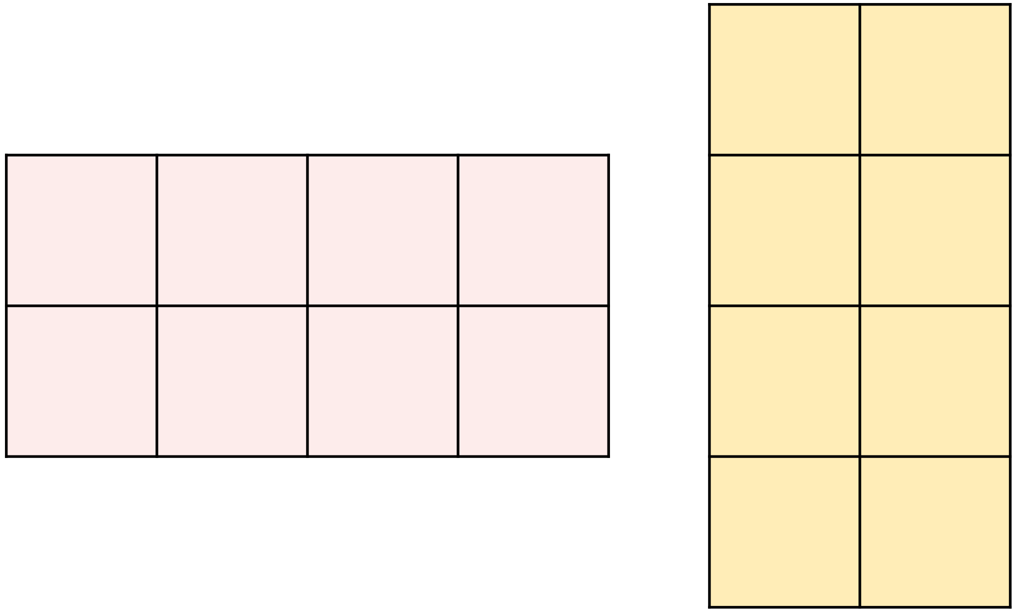
Background: Transformer Architecture

Attention



$A = \mathrm{softmax}(QK^T)$ $\quad$ V

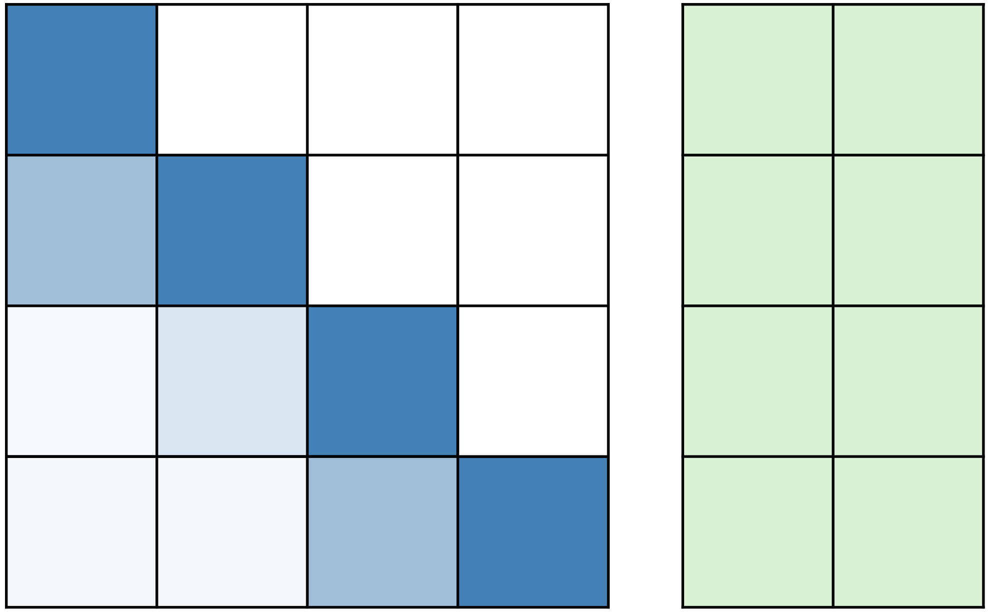
MLP



W_1 $\quad$ W_2

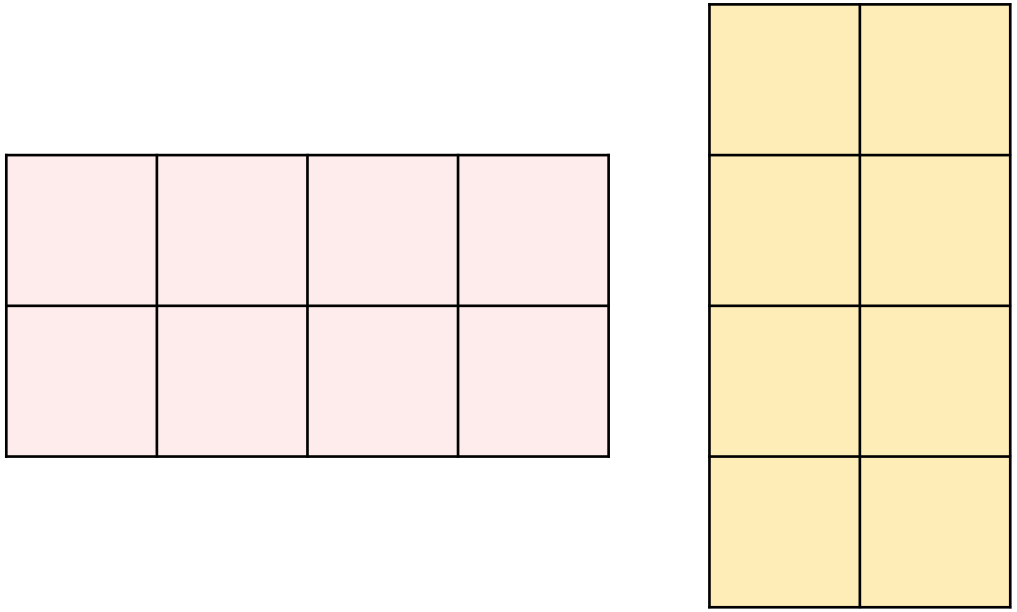
Background: Transformer Architecture

Attention



$$\{W_q, W_k, W_v, W_o\} \in R^{d\text{x}d}$$

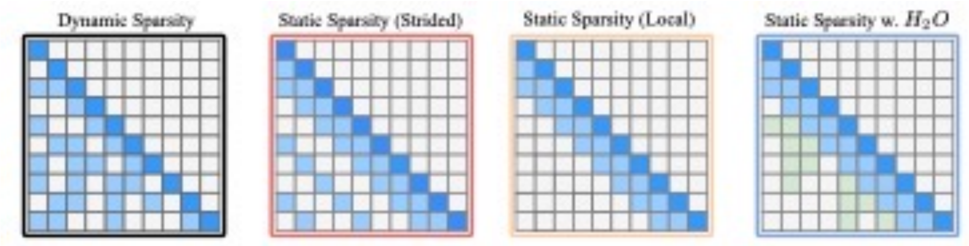
MLP



$$\{W_1, W_2\} \in R^{d\text{x}4d}$$

LLMs are Powerful , but Very Expensive to Deploy

H_2O (*NeurIPS'23*)

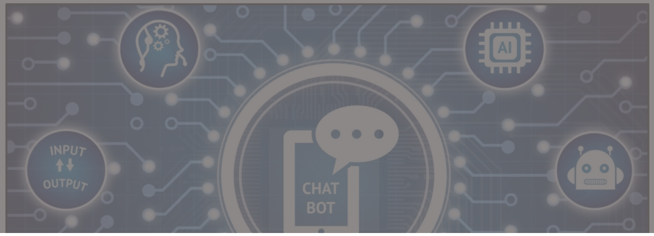


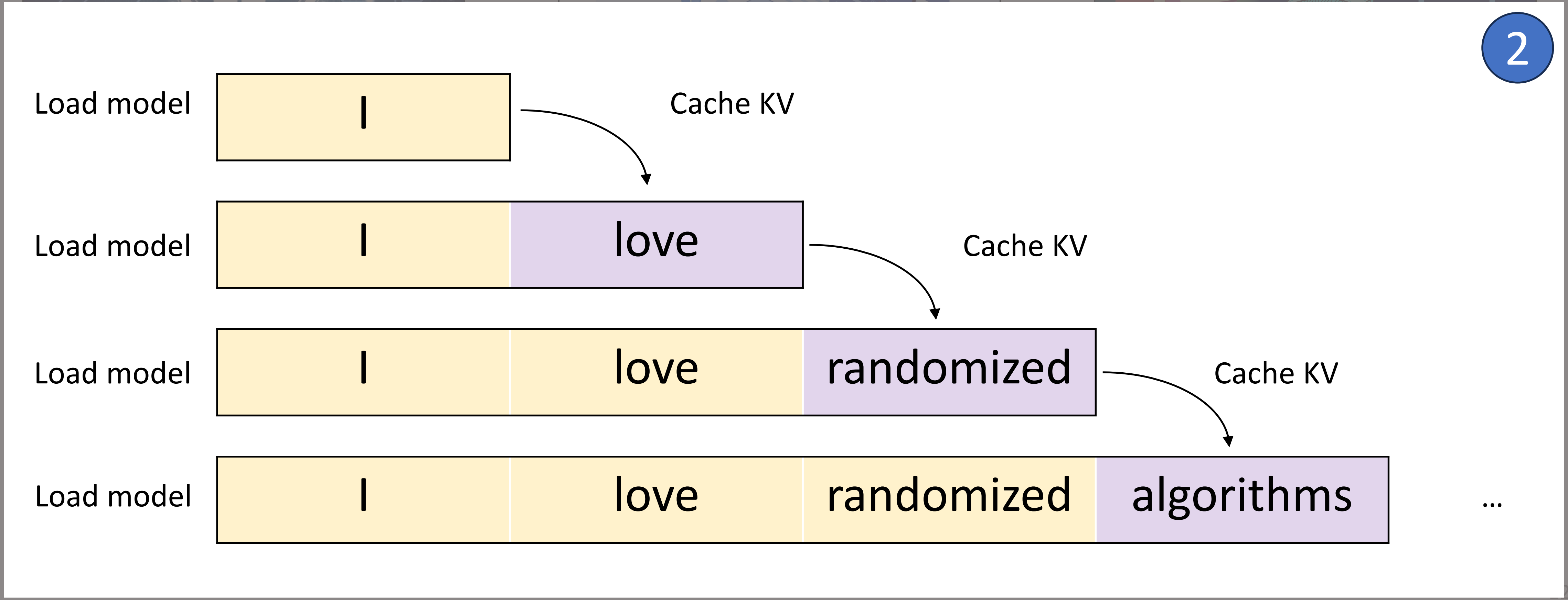
29x, 29x, 3x higher throughput, 1.9x lower latency than DeepSpeed Zero-Inference, HuggingFace Accelerate, and FlexGen with Heavy-Hitter Sparsity.

StreamingLLM (*new* 🔥)

Model 4 million tokens… 22x faster than sliding window recomputation with Attention Sink.

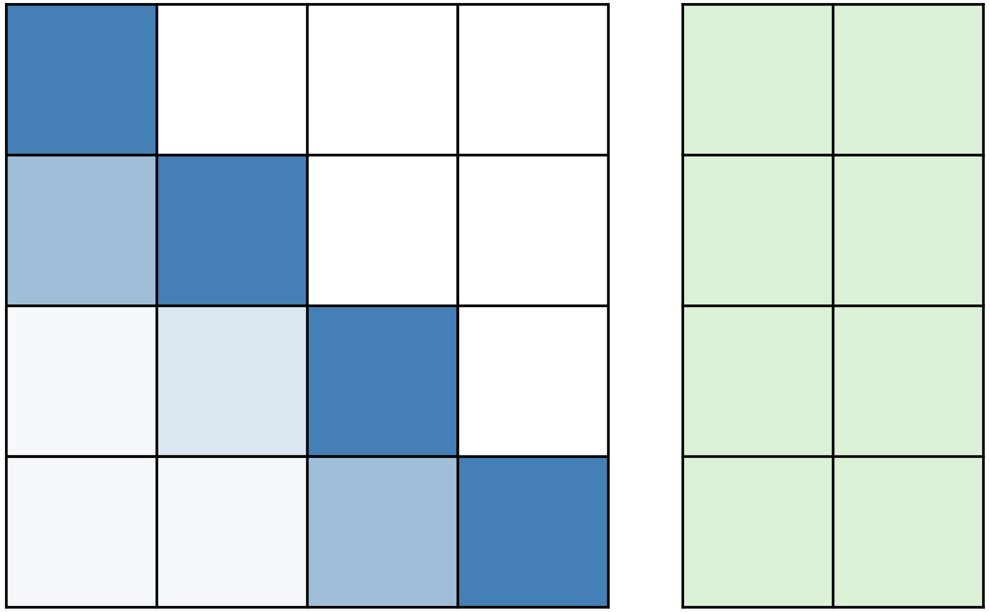
LLMs are Powerful , but Very Expensive to Deploy





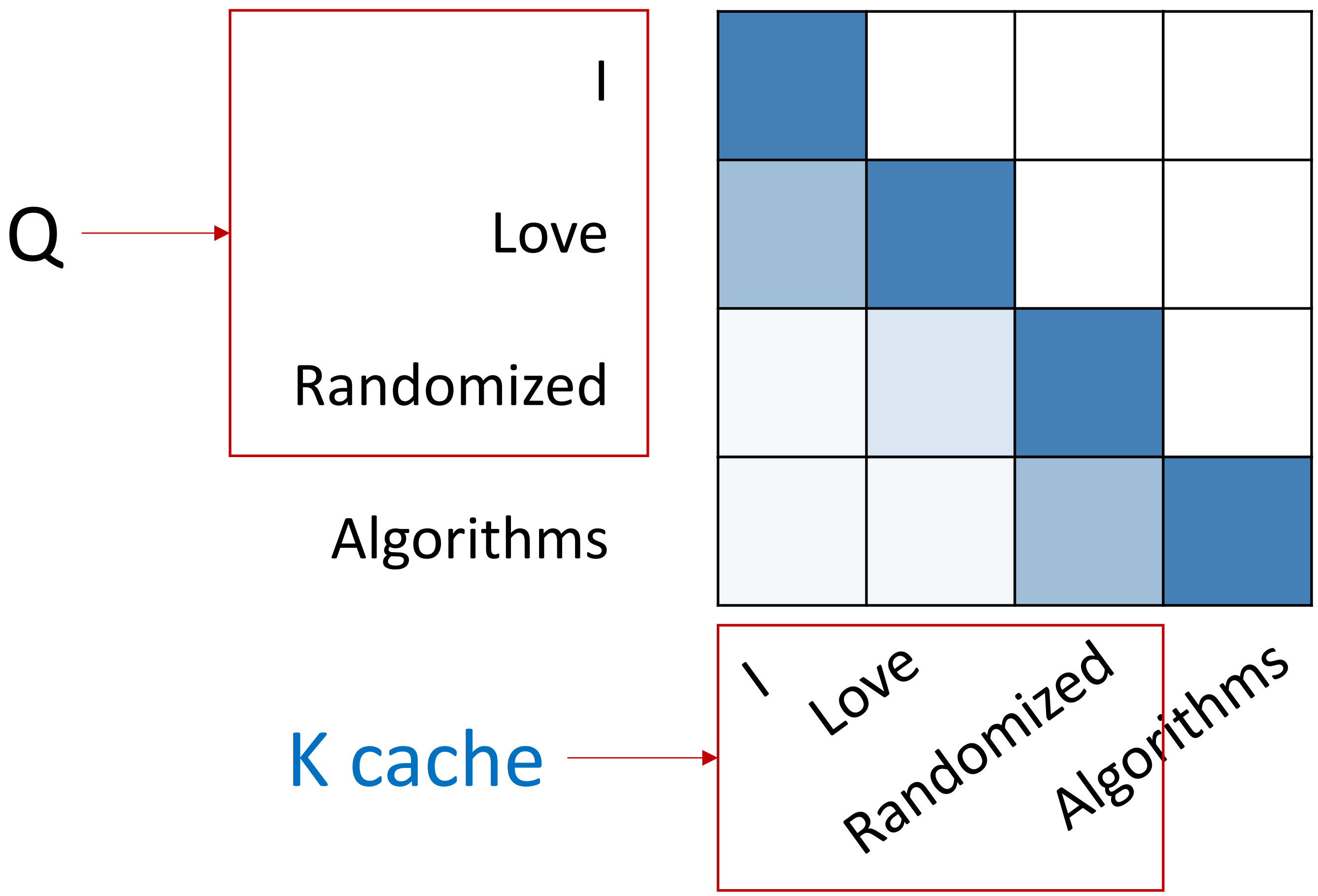
Background: Transformer Architecture

Attention



$A = \mathrm{softmax}(QK^T)$ $\quad V$

KV Cache Bottleneck



KV states for context or previously generated tokens will be cached to avoid re-computation.

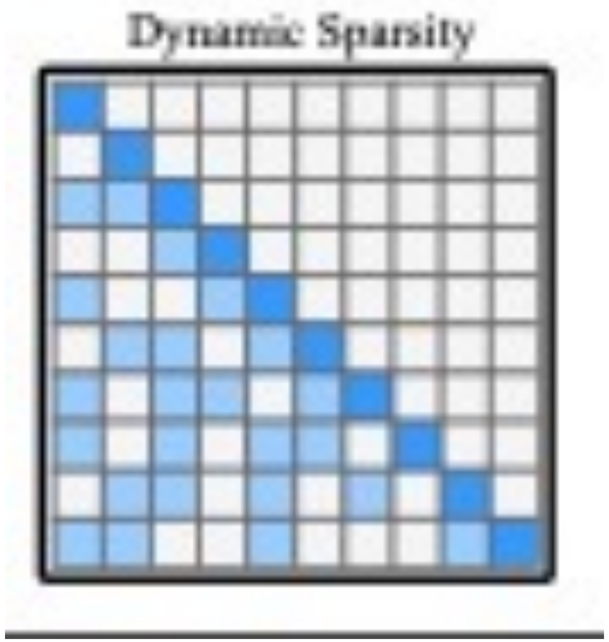
KV cache size scales linearly with sequence length and batch size.

Existing Approaches and Challenges

Naturally, we can limit the cache size like the SW/HW caches. Attention approximation has been widely studied in training long sequences!

But hard to adapt to generation:

- Reduce quadratic attention but not KV cache size
  - e.g., FlashAttention, Reformer

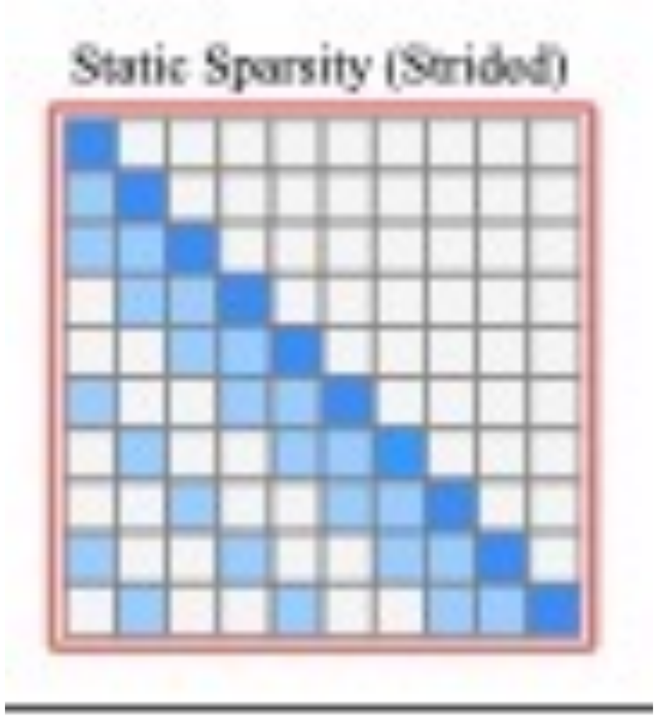


Existing Approaches and Challenges

Naturally, we can limit the cache size like the SW/HW caches. Attention approximation has been widely studied in training long sequences!

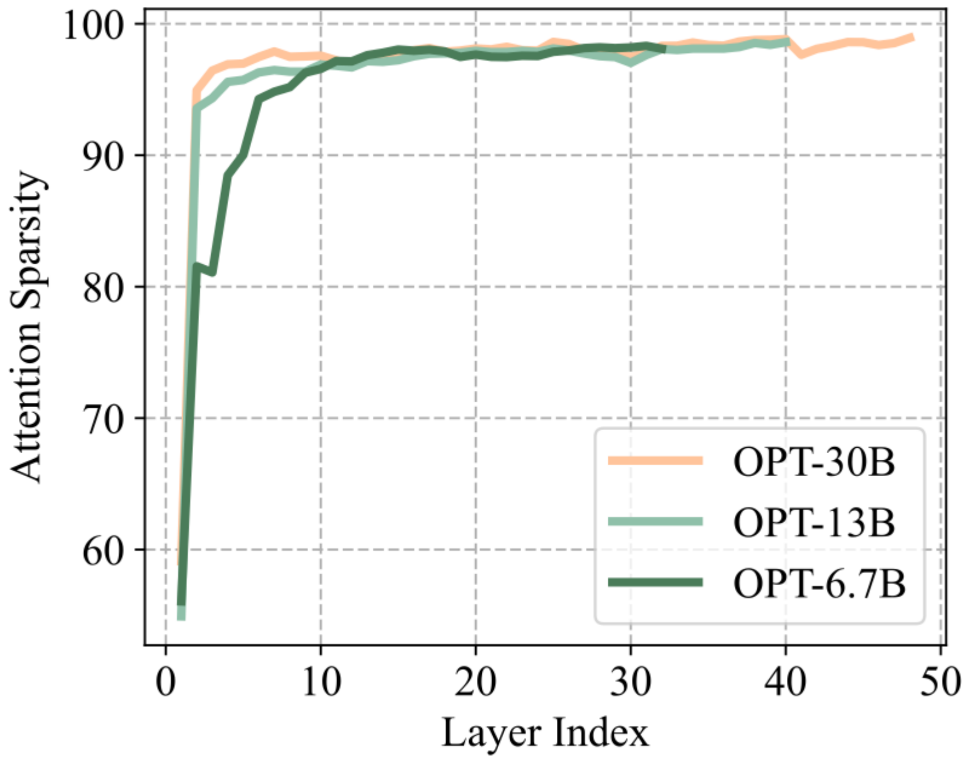
But hard to adapt to generation:

- Reduce quadratic attention but not KV cache size
  - e.g., FlashAttention, Reformer
- Result high cache miss rates and degrade accuracy
  - e.g., Sparse Transformer
- Expensive eviction policy
  - e.g., Gisting Tokens



An ideal cache has a small cache size, a low miss rate, and a low-cost eviction policy.

Sparsity for Smaller Cache Size



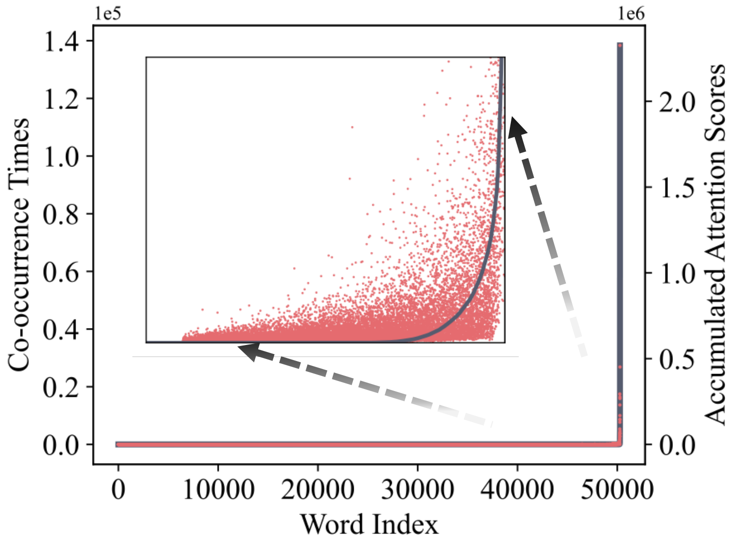
Observation: although densely trained, LLMs

- attention score matrices are highly sparse, with a sparsity over 95% in almost all layers
- leads to 20× potential KV cache reduction
- maintains same accuracy

Attention sparsity widely exists in pre-trained models, e.g. OPT /LLaMA /Bloom/GPT.

Heavy-Hitters for Low Miss Rate

Challenge: how to evict tokens? Once evicted, future tokens can no longer attend to it

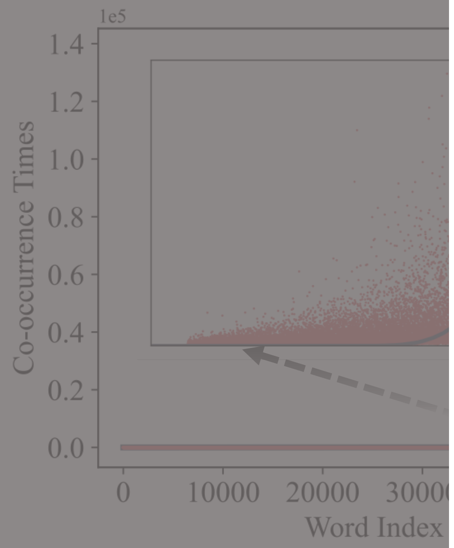


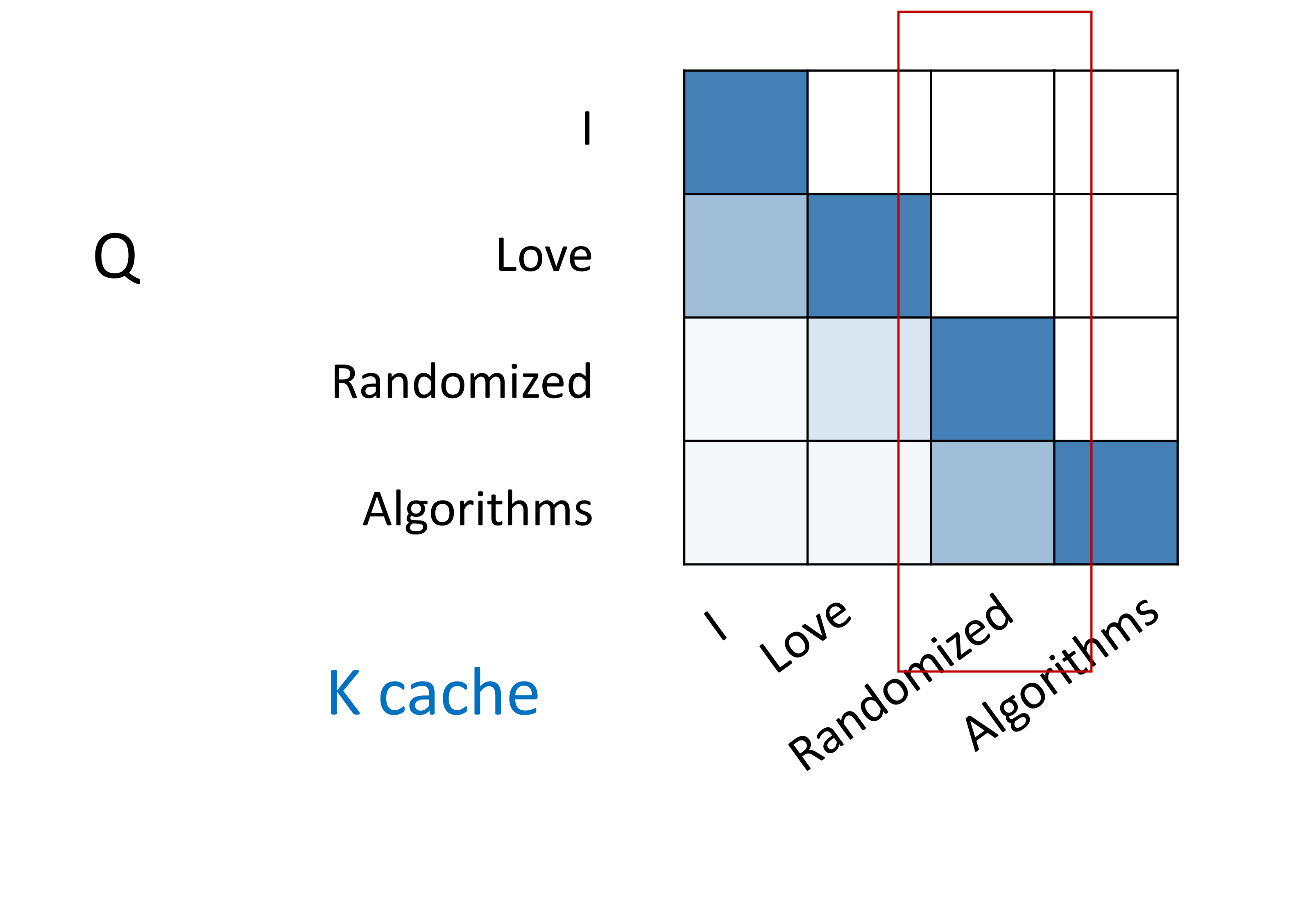
Key Observation: a small set of tokens are important along the generation

- accumulated attention scores of all the tokens follow a power-law distribution

Heavy-Hitters for Low Miss Rate

Challenge: how to evict to



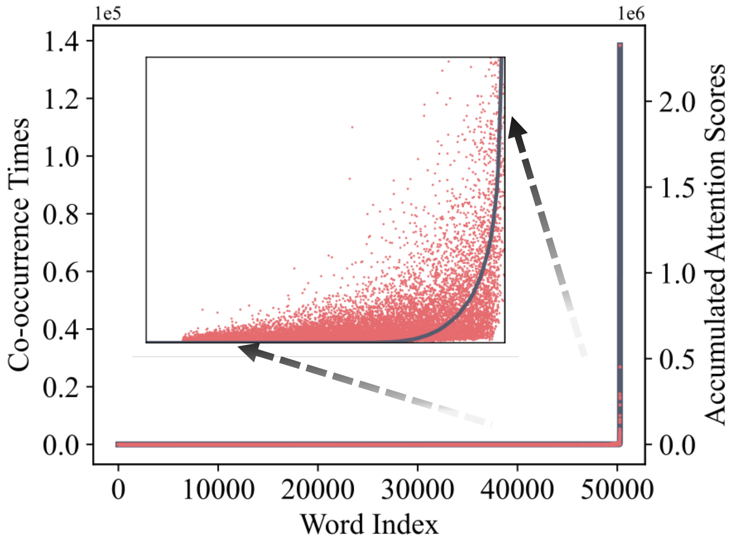


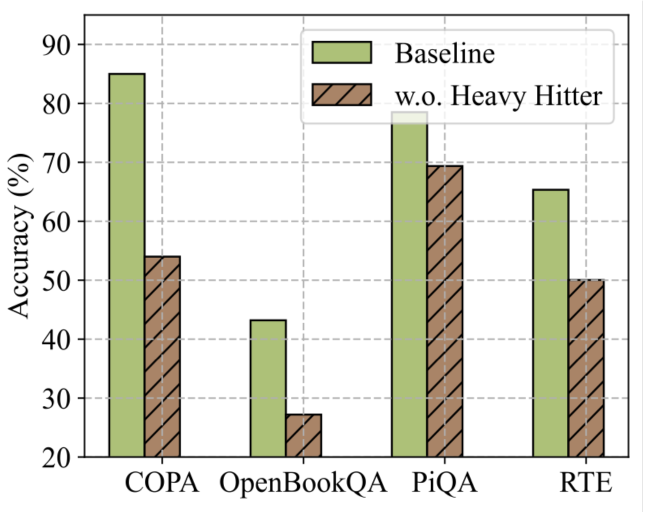
Key Observation: a small s

- accumulated attenti

Heavy-Hitters for Low Miss Rate

Challenge: how to evict tokens? Once evicted, future tokens can no longer attend to it



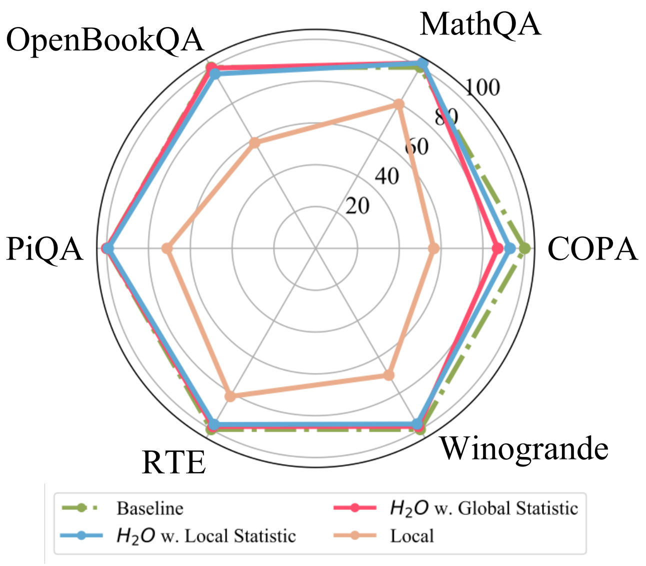


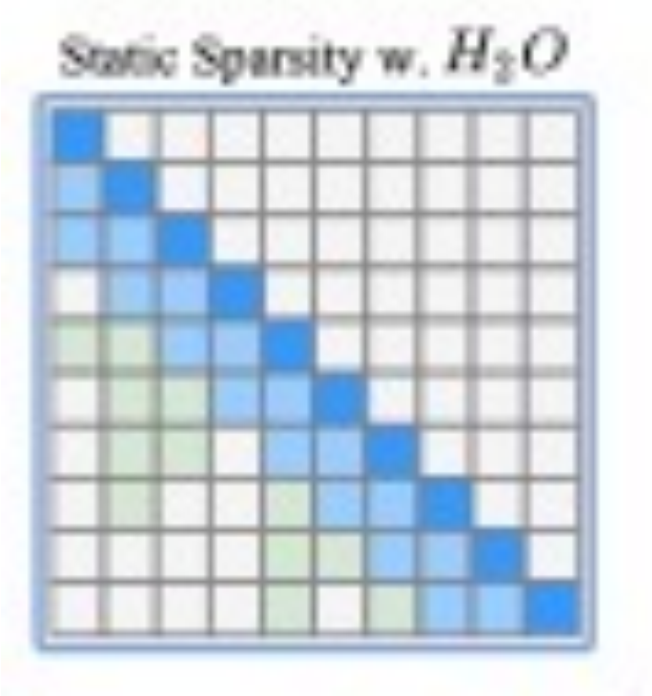
Key Observation: a small set of tokens are important along the generation

- accumulated attention scores of all the tokens follow a power-law distribution
- masking heavy-hitter tokens degrades model quality

Greedy Algorithm for Low-cost Policy

Challenge: how to deploy such algorithm without access to the full attention?

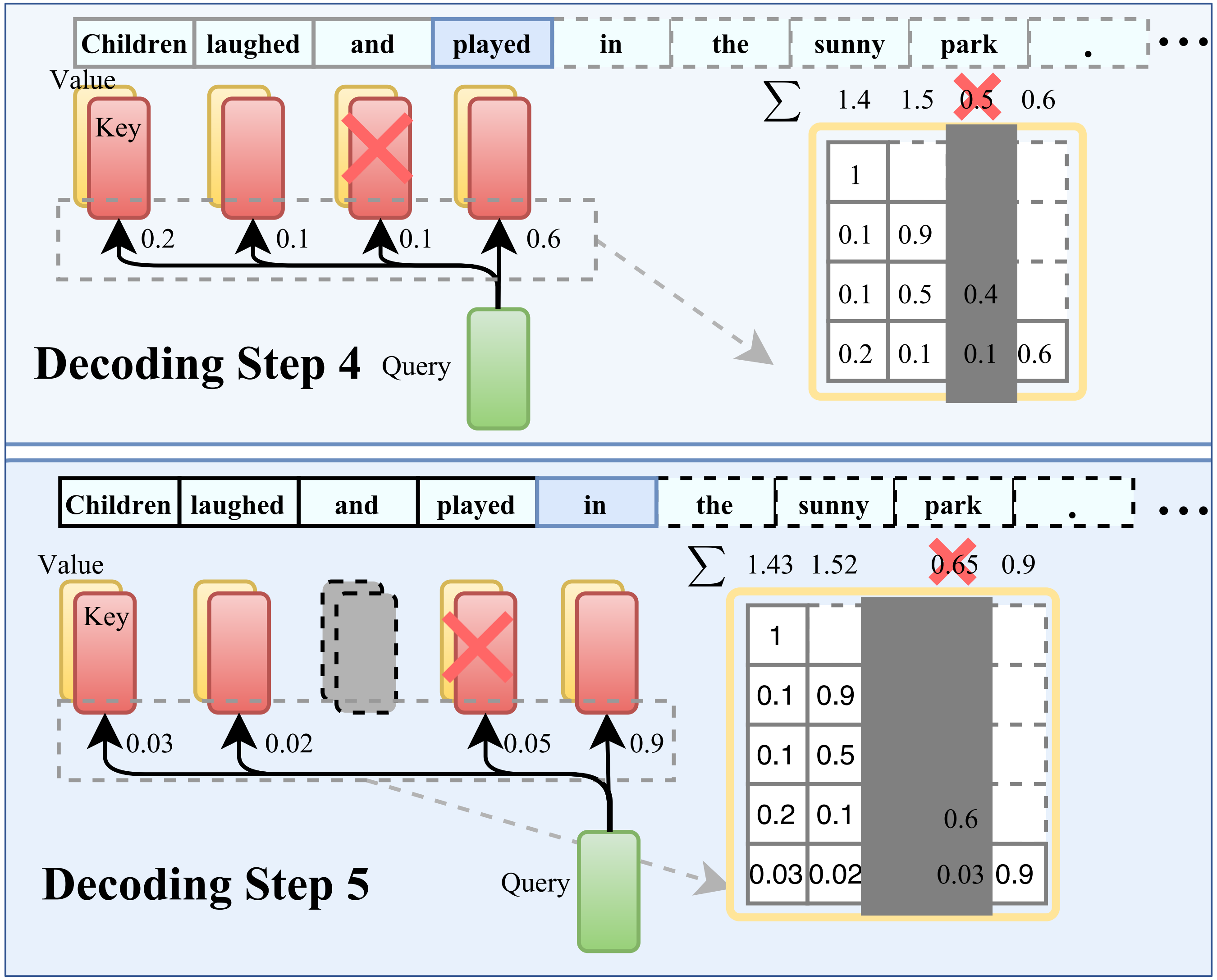




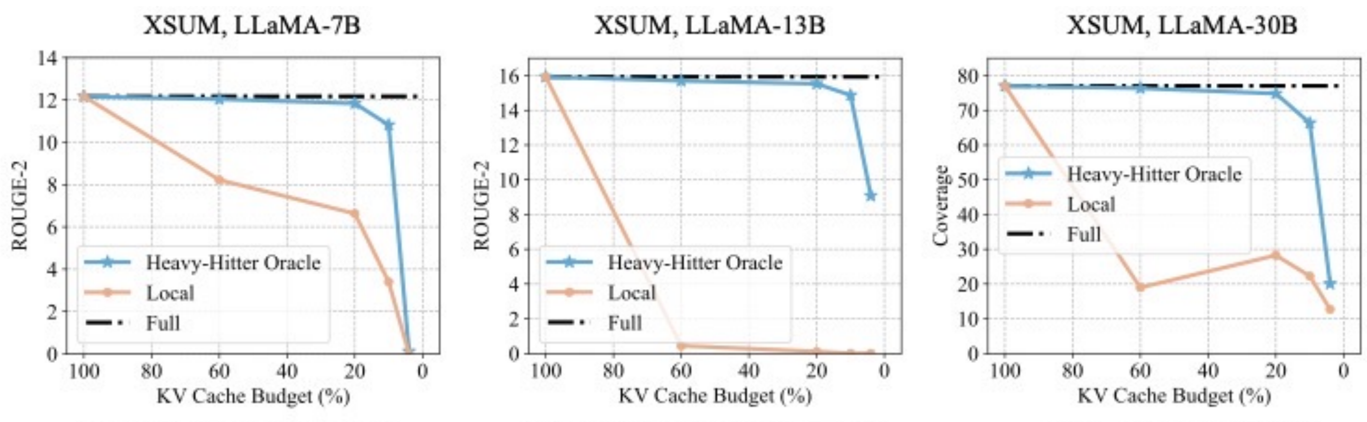
Idea: local greedy algorithm

- sum up the attention scores of the previous tokens every decoding step
- Add local / recent tokens

H_2O: Heavy Hitter Oracle



H_2O: 3-29X Throughput and 1.9X Latency



A100	FlexGen	H_2O
Throughput (token/s)	494	918 (1.9X)
Latency (s)	99	53 (1.9X)

	Hugging Face	Deep Speed	FlexGen	H_2O
Throughput (T4) token/s	0.6	0.6	8.5	18.83 (3-29X)

- compatible with quantization
- generate sentences with fewer repeated words and more creativity

Model Input

In a small, bustling cafe nestled in the heart of a vibrant city, a serendipitous event unfolded, leaving a lasting impression on all who witnessed it. As the patrons sat sipping their coffees and engaging in animated conversations, a talented street musician entered the cafe, carrying a weathered guitar and radiating an aura of creativity.

LLaMA-7B Full Cache Output

He began to play, and the patrons were captivated. The musician's performance was so moving that the patrons began to applaud, and the musician was so moved that he began to cry. The patrons were so moved that they began to cry, and the musician was so

LLaMA-7B Local 20% Cache Output

He ((((((((((((((, [)), 1999, 2000, 2001, 1, and, and, and, and, and, and, and, and, and, and, and,

LLaMA-7B H_2O 20% Cache Output

He began to play, and the room was filled with the sound of his music. The patrons of the cafe were enthralled by the music, and the atmosphere was electric. The cafe was packed with people, all of whom were enjoying the music. The musician was a young

Model Input

ling cafe nestled in the heart of a vibrant city, a serendipitous event unfolded, leaving a lasting
all who witnessed it. As the patrons sat sipping their coffees and engaging in animated conversations,
t musician entered the cafe, carrying a weathered guitar and radiating an aura of creativity.

What are these heavy hitters?

He began to play, and the patrons were captivated. The musician's performance was so moving that the patrons began to applaud, and the musician was so moved that he began to cry. The patrons were so moved that they began to cry, and the musician was so

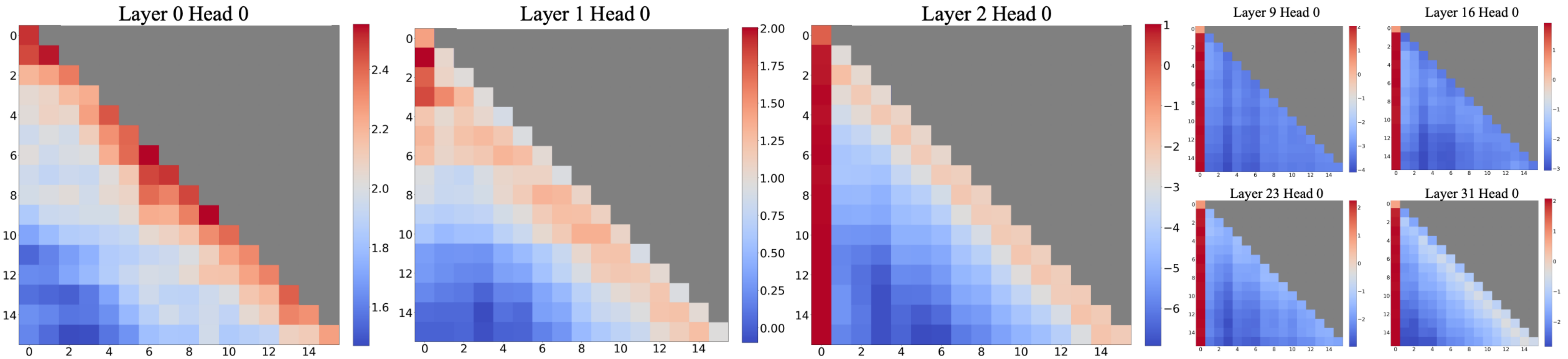
LLaMA-7B Local 20% Cache Output

He ((((((((((((((, [)), 1999, 2000, 2001, 1, and, and, and, and, and, and, and, and, and, and, and,

LLaMA-7B H_2O 20% Cache Output

He began to play, and the room was filled with the sound of his music. The patrons of the cafe were enthralled by the music, and the atmosphere was electric. The cafe was packed with people, all of whom were enjoying the music. The musician was a young

Phenomenon: Attention Sink



Average attention logits in Llama-2-7B over 256 sentences

First few tokens!

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

Understanding Attention Sinks

- SoftMax operation's role in creating attention sinks — attention scores have to sum up to one for all contextual tokens. (*SoftMax-Off-by-One, Miller et al. 2023*)

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

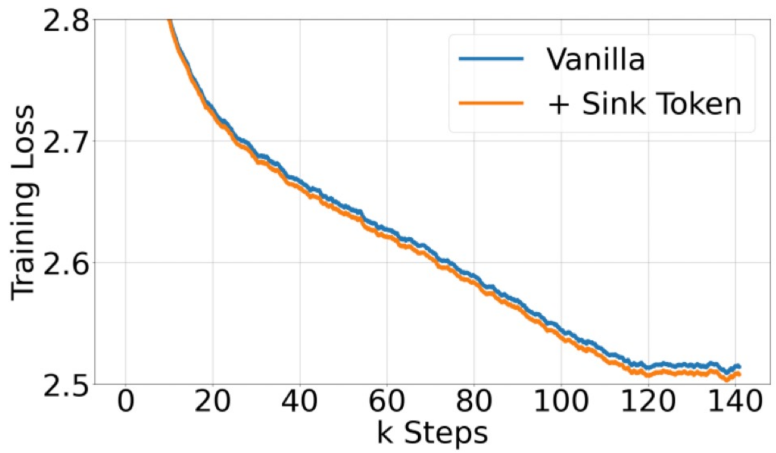
- Initial tokens' advantage in becoming sinks due to their visibility to subsequent tokens, rooted in autoregressive language modeling.

- The model learns a bias towards their absolute position rather than the semantics are crucial.

Llama-2-13B	PPL (↓)
0+1024 (window)	5158.07
4+1024	5.40
4"\n"+1020	5.6

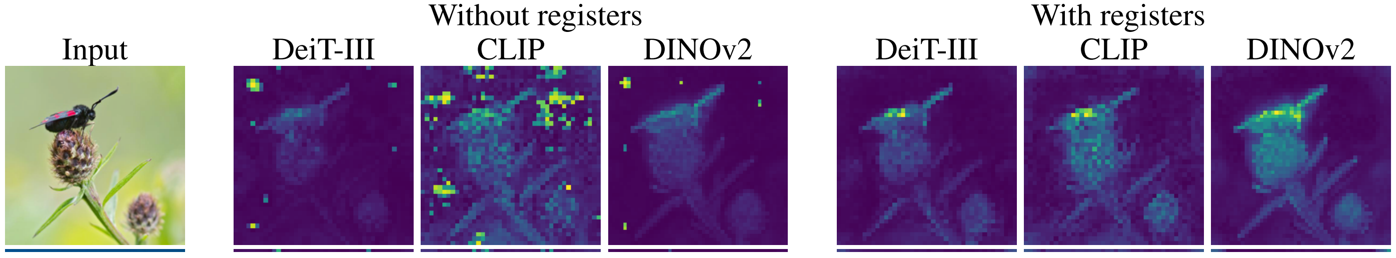
Understanding Attention Sinks

- Pre-train with a Dedicated Attention Sink Token

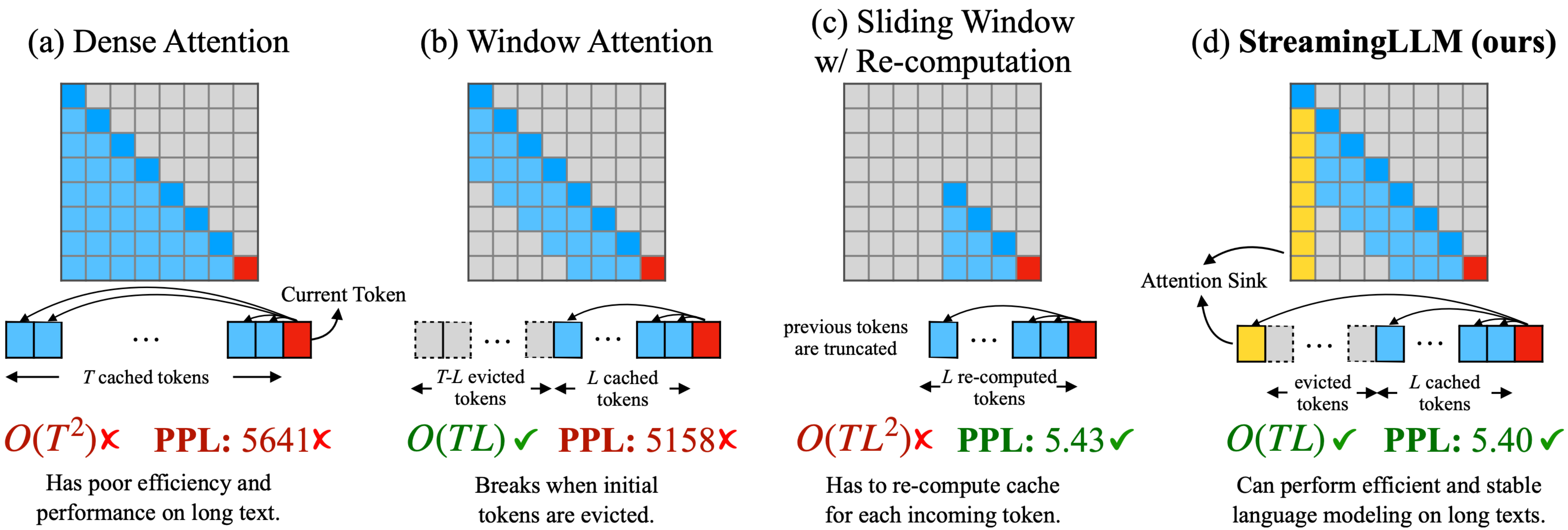


Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	**18.01**	18.01	18.02

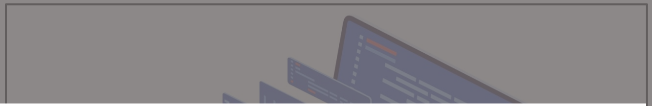
- Similar Phenomenon in *Darcet et al. Vision transformers need registers*

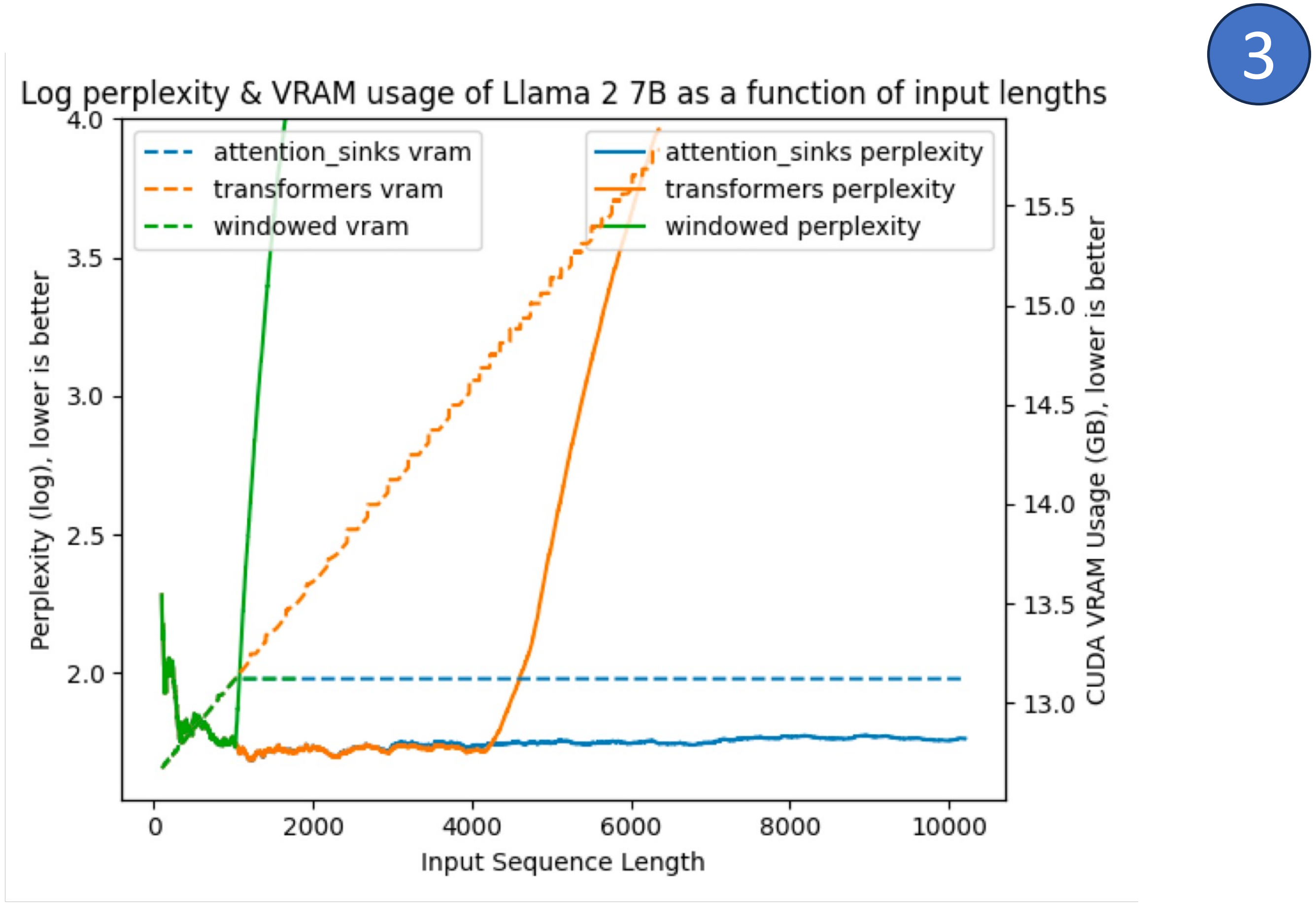


StreamingLLM



LLMs are Powerful , but Very Expensive to Deploy





AI Agents

en + 64 batch size)

oad model, KV cache 100 times

Infinite Streaming Ability

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

Key challenge:

- Pre-trained model (e.g., LLaMA) cannot go beyond its pre-trained context window

Train:

1	2	3	4	5	6	7	8

Test:

1	2	3	4	5	6	7	8	?	?

Opportunity with StreamingLLM:

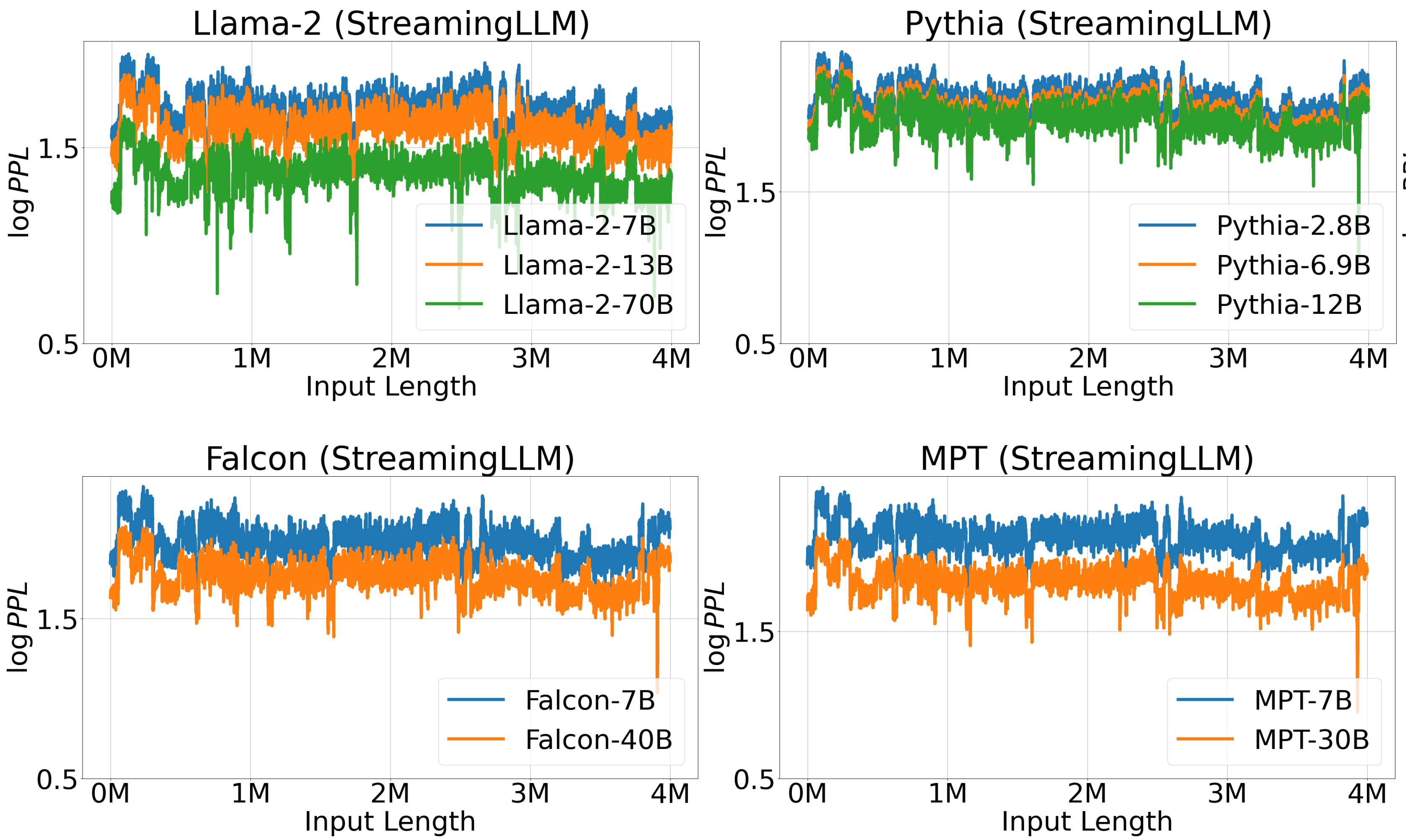
Train:

1	2	3	4	5	6	7	8

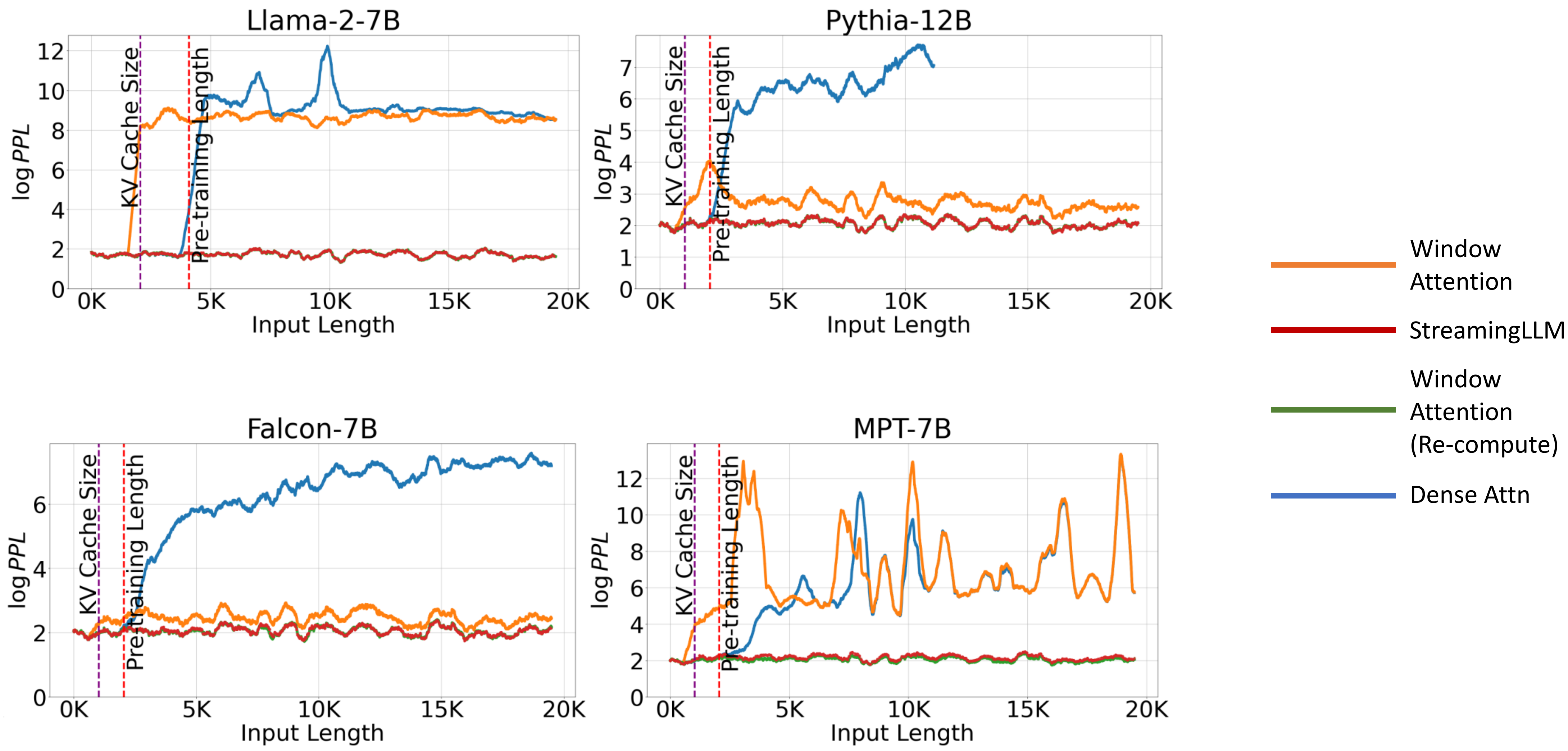
Test:

1	2	3	4	x	x	5	6	7	8

Stably Model up to 4 Million Tokens



22X Faster than Sliding Window Recomputation



Infinite Streaming Ability

Urge LLMs in streaming applications such as multi-round dialogues, where long intera needed.

But StreamingLLM will forget the middle contents?

Key challenge:

- Pre-trained model (e.g., LLaMA) cannot go beyond its pre-trained context window

Train: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Test: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ? | ? |

Opportunity with StreamingLLM:

Train: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Test: | 1 | 2 | 3 | 4 | x | x | 5 | 6 | 7 | 8 |

The perplexity remains stable throughout up to 4 Million Tokens!

StreamingH2O: Infinite Streaming Ability

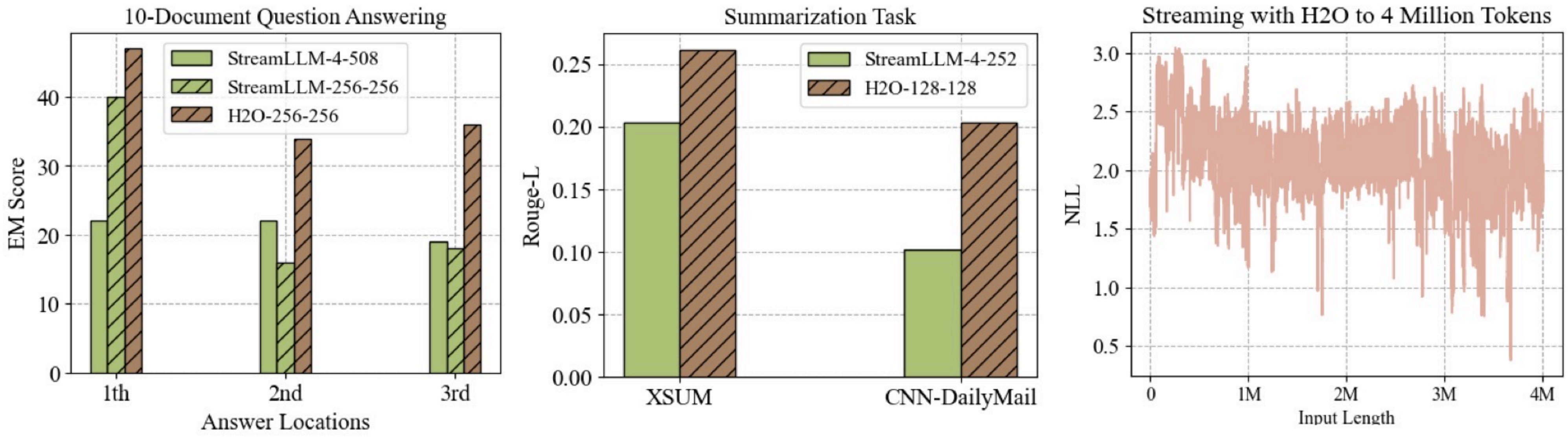
Similar position squeezing can be deployed on H2O

Train:

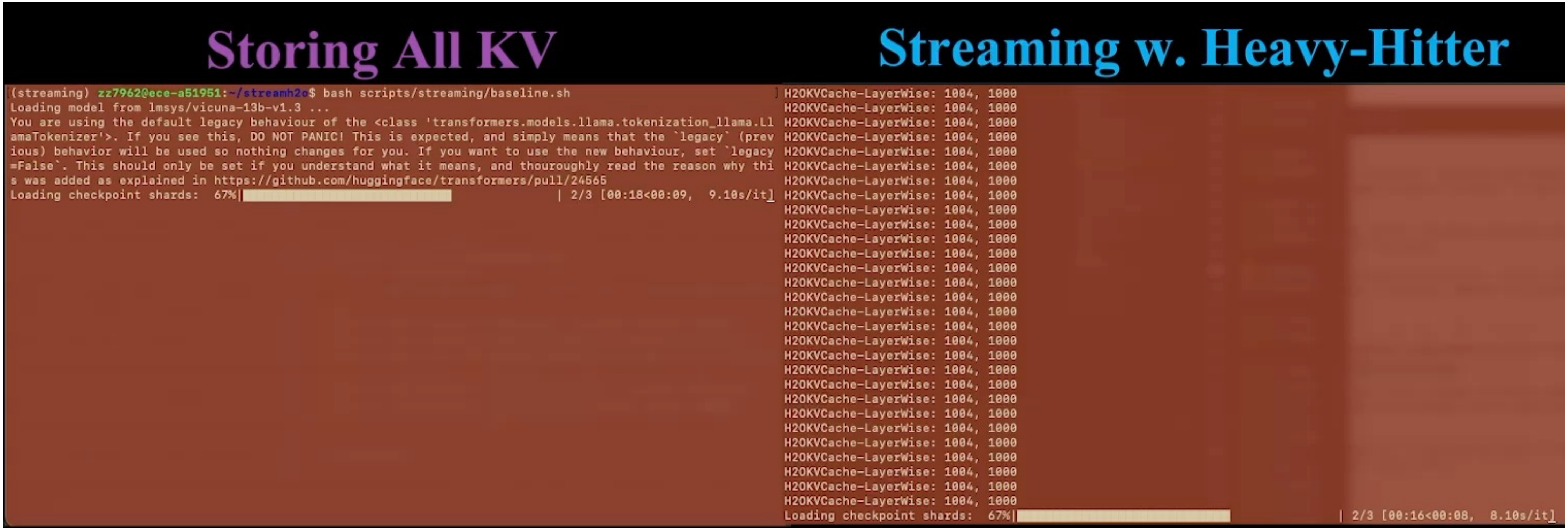
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Test:

1	**x**	**2**	**x**	**3**	**4**	**5**	**6**	**7**	**8**

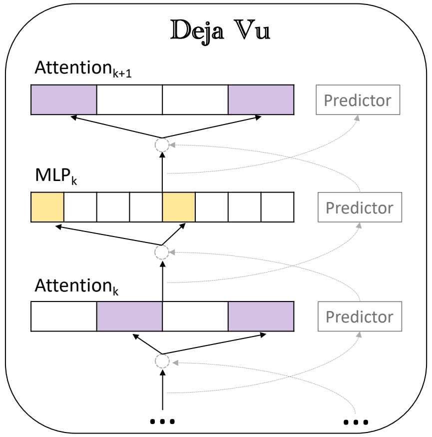


Demo


Storing All KV
Streaming w. Heavy-Hitter
(streaming) zz7962@ece-a51951:~/streamh2o$ bash scripts/streaming/baseline.sh
Loading model from lmsys/vicuna-13b-v1.3 ...
You are using the default legacy behaviour of the <class 'transformers.models.llama.tokenization_llama.LlamaTokenizer'>. If you see this, DO NOT PANIC! This is expected, and simply means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to use the new behaviour, set `legacy=False`. This should only be set if you understand what it means, and thouroughly read the reason why this was added as explained in https://github.com/huggingface/transformers/pull/24565
Loading checkpoint shards: 67%| | 2/3 [00:18<00:09, 9.10s/it]
H2OKVCache-LayerWise: 1004, 1000
Loading checkpoint shards: 67%| | 2/3 [00:16<00:08, 8.10s/it]

LLMs are Powerful , but Very Expensive to Deploy

Deja Vu *(ICML'23)*

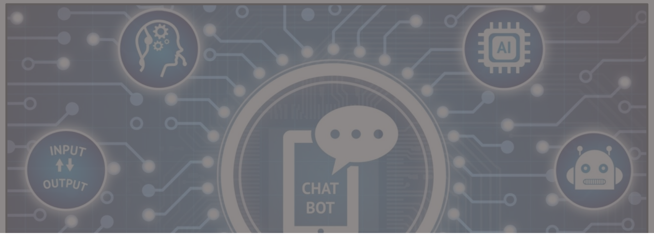


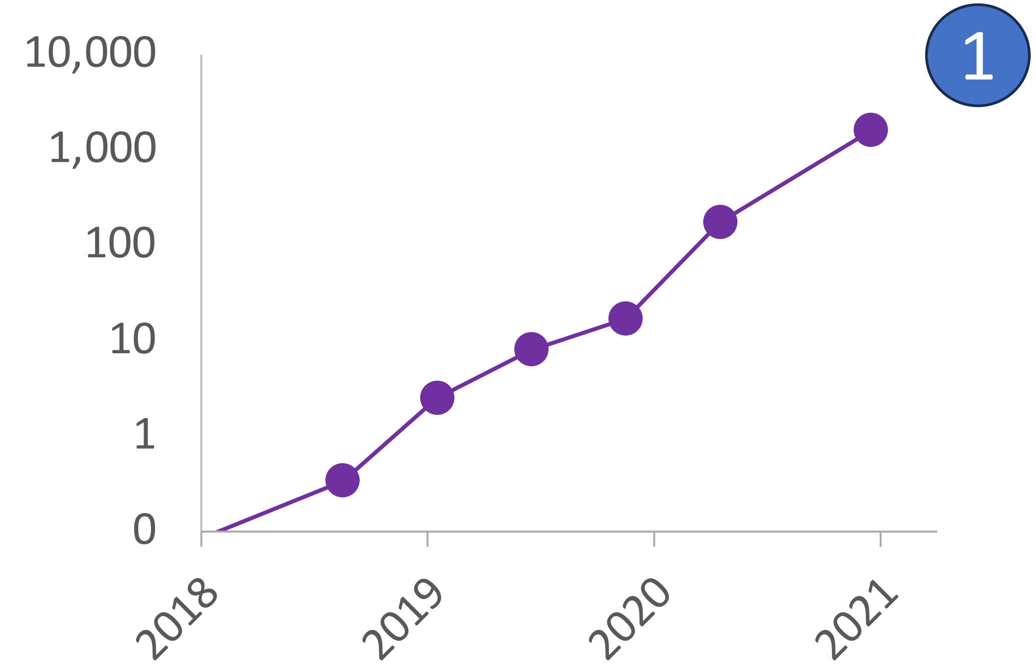
2x lower latency than FasterTransformer and 6x than HuggingFace on 8xA100 with contextual sparsity.

Compress, Then Prompt *(new 🔥)*

8x extreme model compression (Sparse+Quantize) with Prompt Recovery.

LLMs are Powerful , but Very Expensive to Deploy

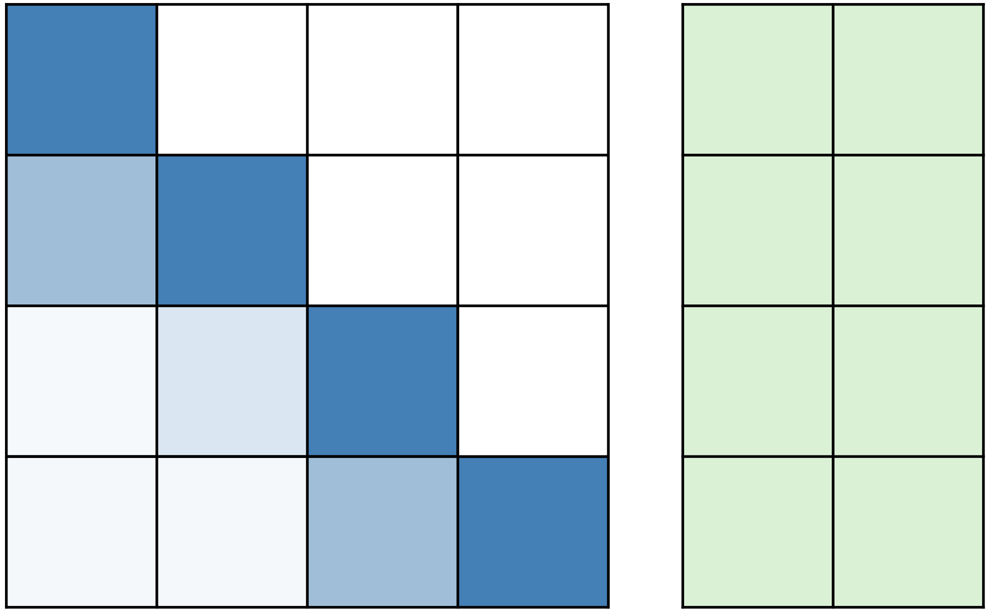




Exponential model size

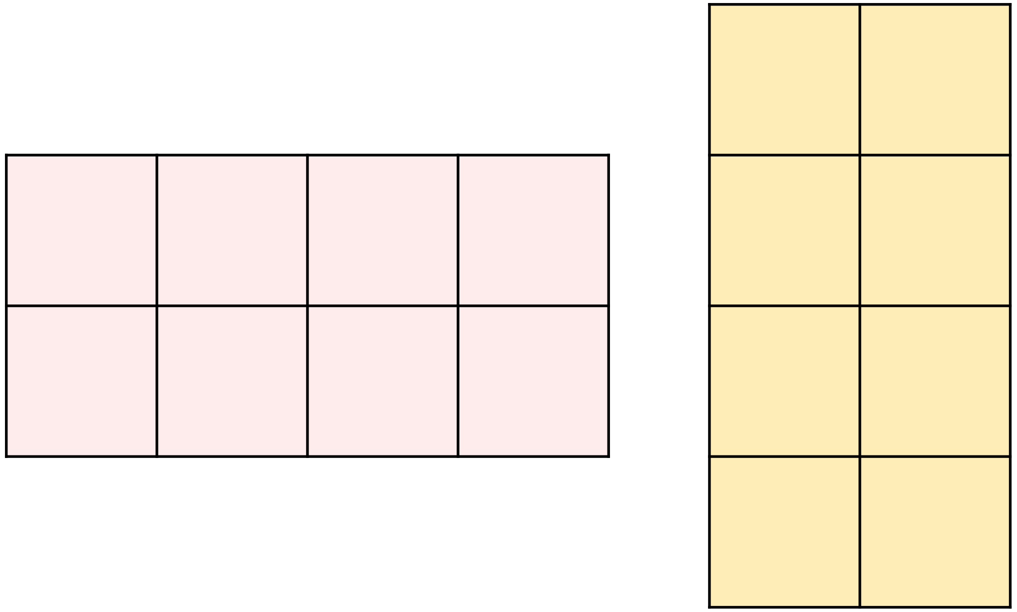
Background: Transformer Architecture

Attention



$$\{W_q, W_k, W_v, W_o\} \in R^{d\mathrm{x}d}$$

MLP



$$\{W_1, W_2\} \in R^{d\mathrm{x}4d}$$

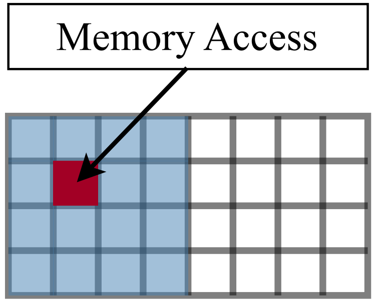
Existing Approaches and Challenges

The idea sparsity or pruning is not new!

- Long history in ML, statistics, neuroscience, signal processing … *(Lecun et al. 90, Donoho 92, Tibshirani 96, Foldiak et al. 03, Candes et al. 05)*

But hard to speed up sparse LLMs in wall-clock time and maintain quality

- Expensive and infeasible to finetune or retrain
- Difficult to find sparsity that preserves emergent ability of LLMs
- Unstructured sparsity is not hardware-efficient *(Hooker et al. 20)*

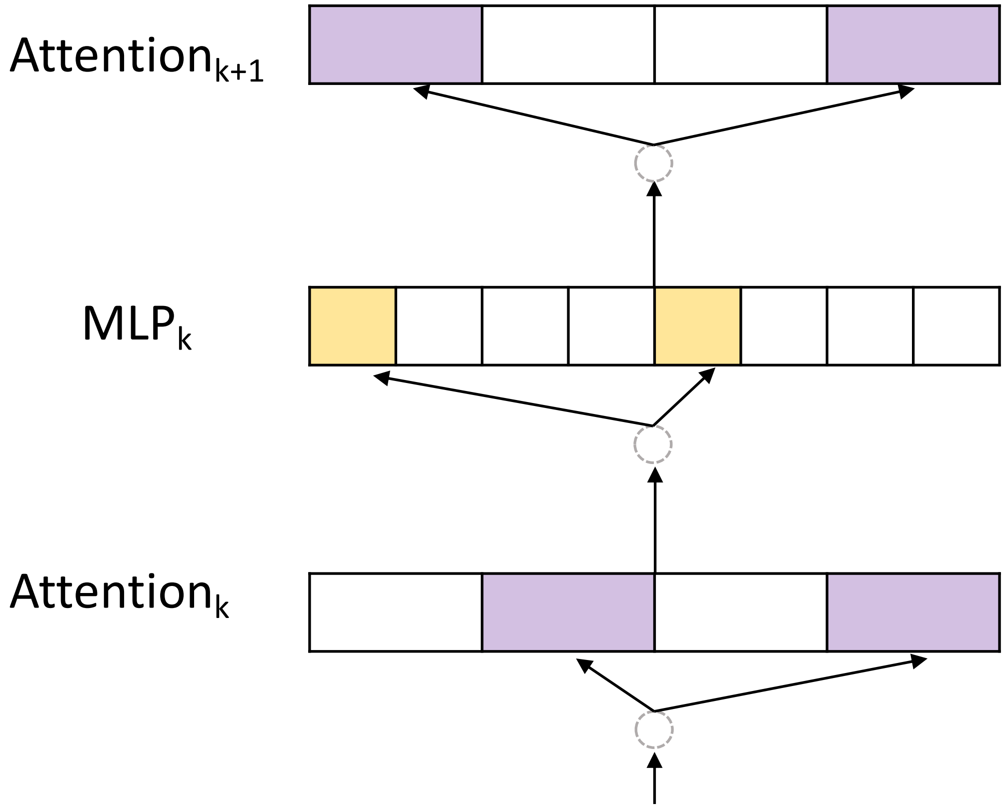


~~Ideal~~ sparsity requires no retraining, maintains quality, and speeds up in wall-clock time.

Contextual

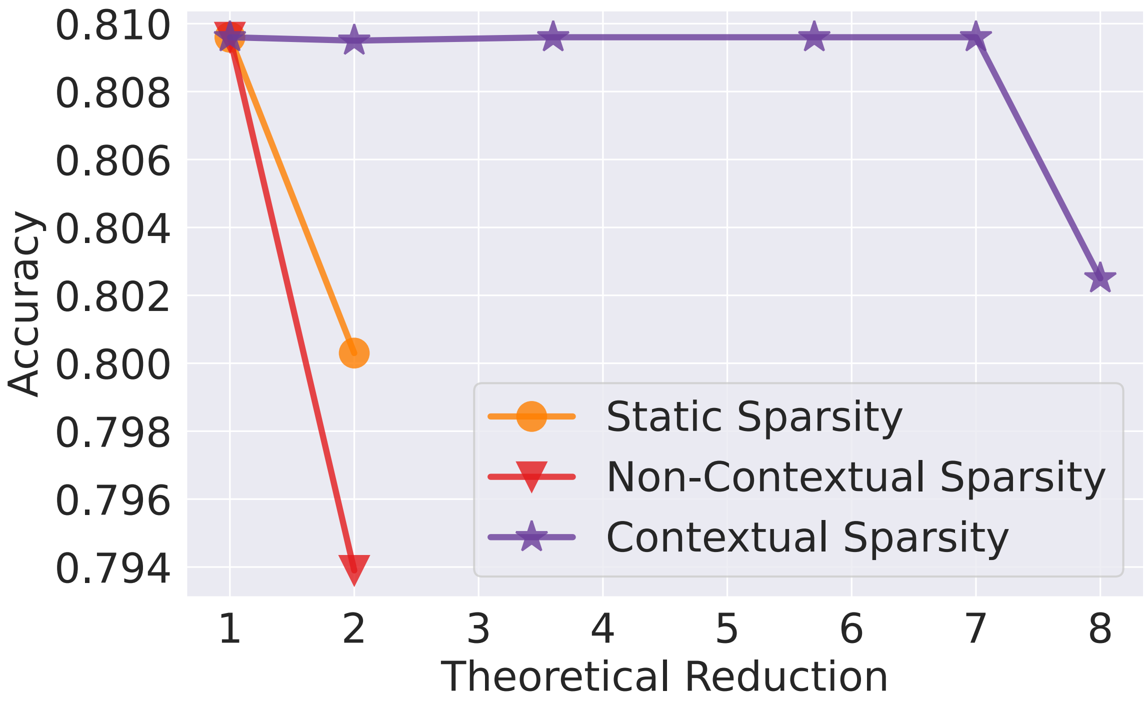
Hypothesis: Contextual Sparsity Exists Given Any Input

Contextual sparsity: small, input-dependent sets of **attention** heads and **MLP** parameters that lead to (approximately) the same output as the full model for an input.



Inspired by:
connections between LLMs, Hidden Markov Models and Viterbi algorithm (Xie et al.)

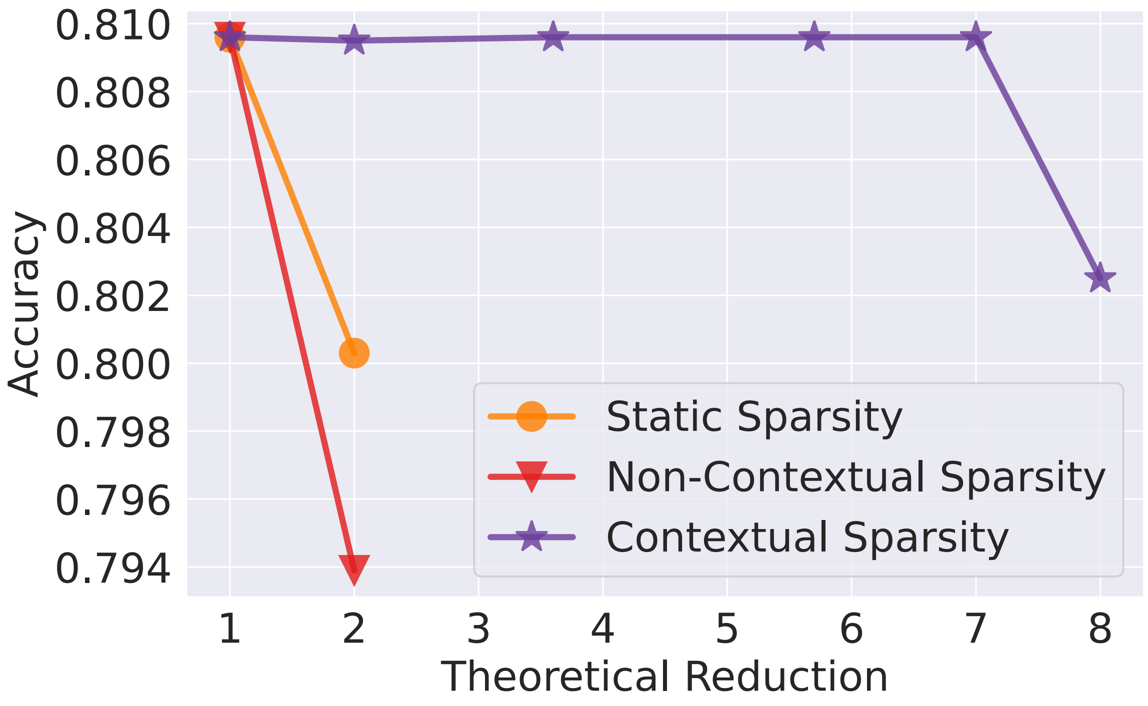
Contextual Sparsity: Existence

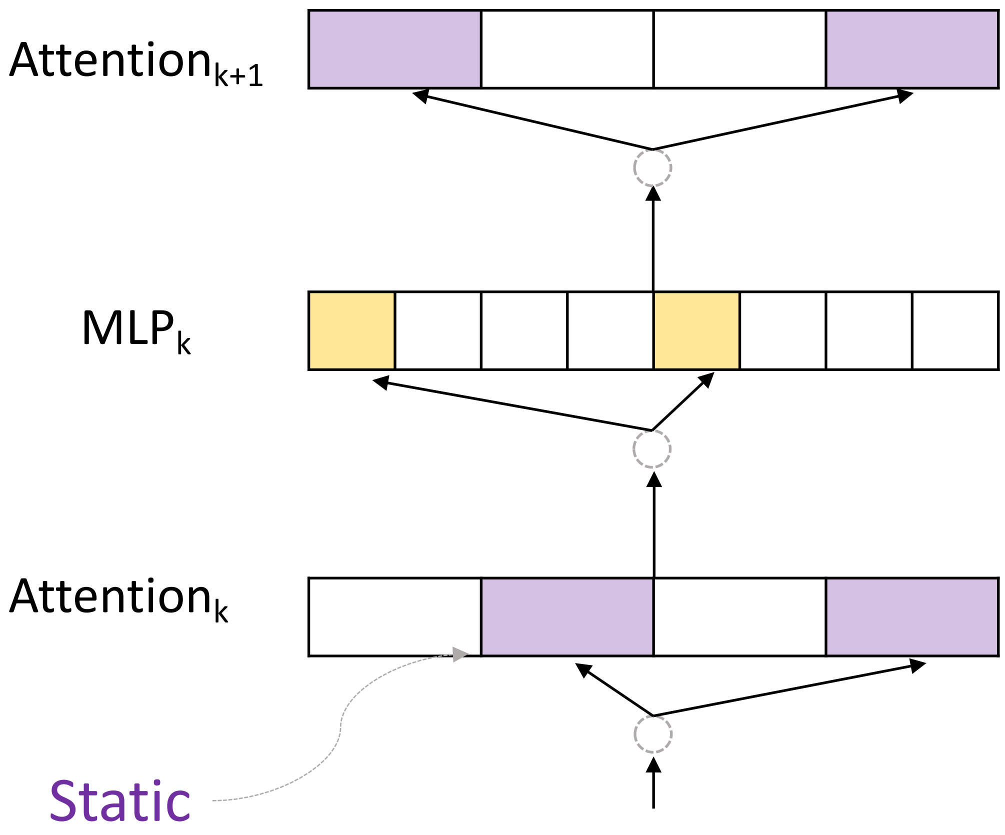


Observation:
keep only high activation in attention/MLP blocks

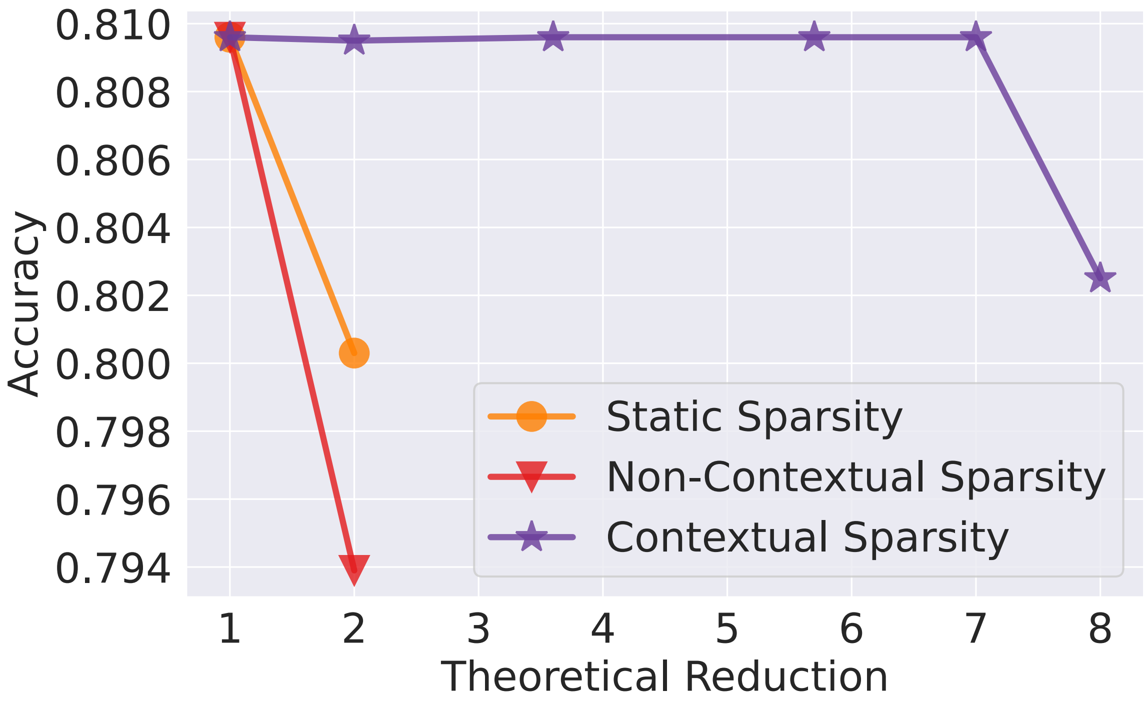
- 85% structured sparse
  - 80% attention, 95% MLP
- lead to 7× potential parameter reduction for each input
- maintain accuracy

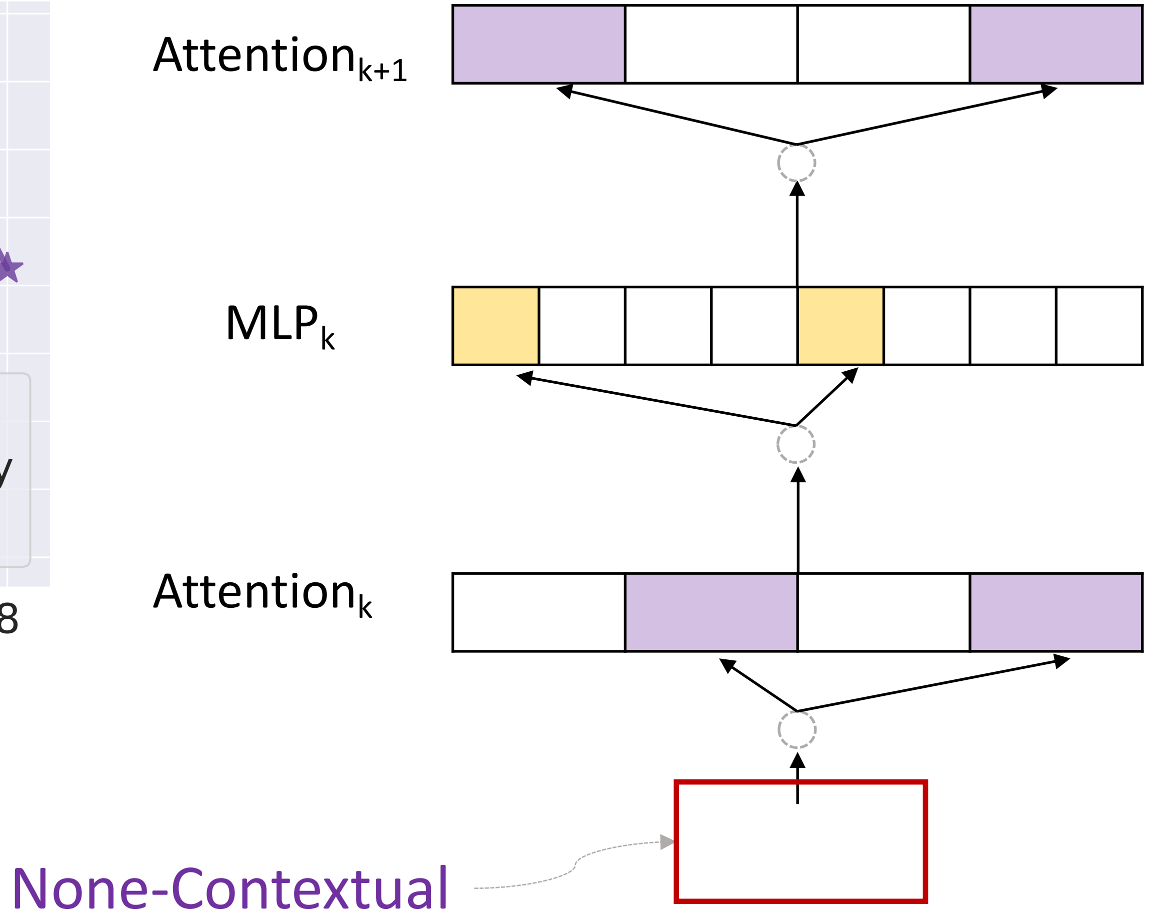
Contextual Sparsity: Existence



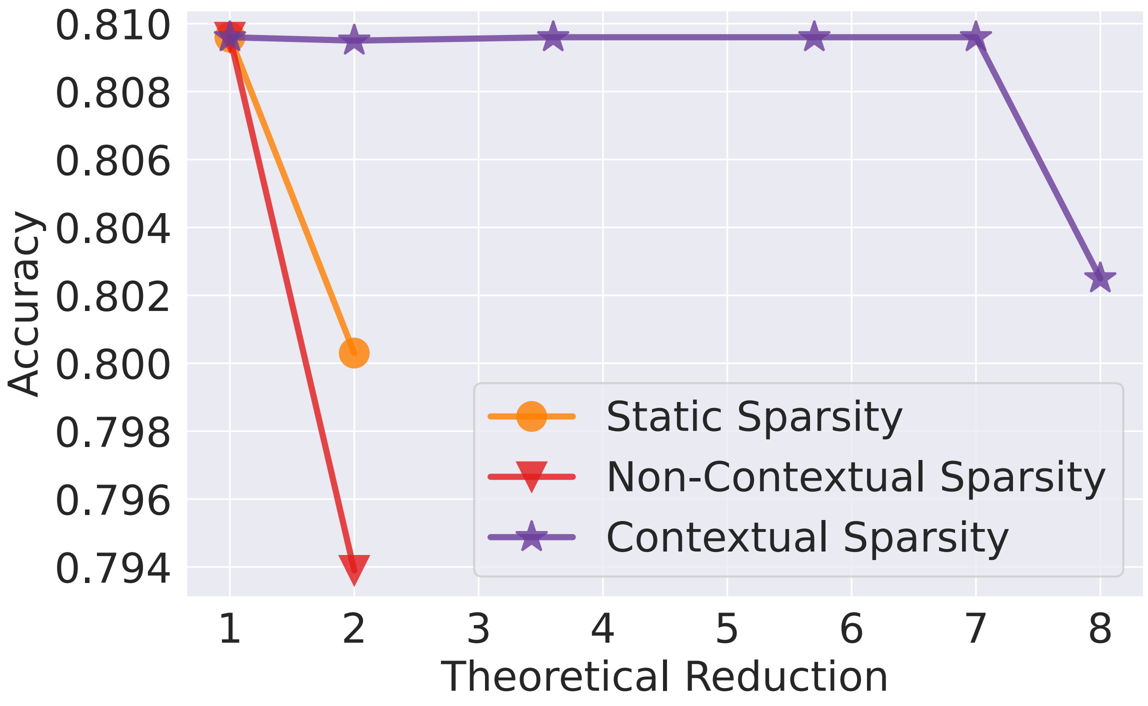


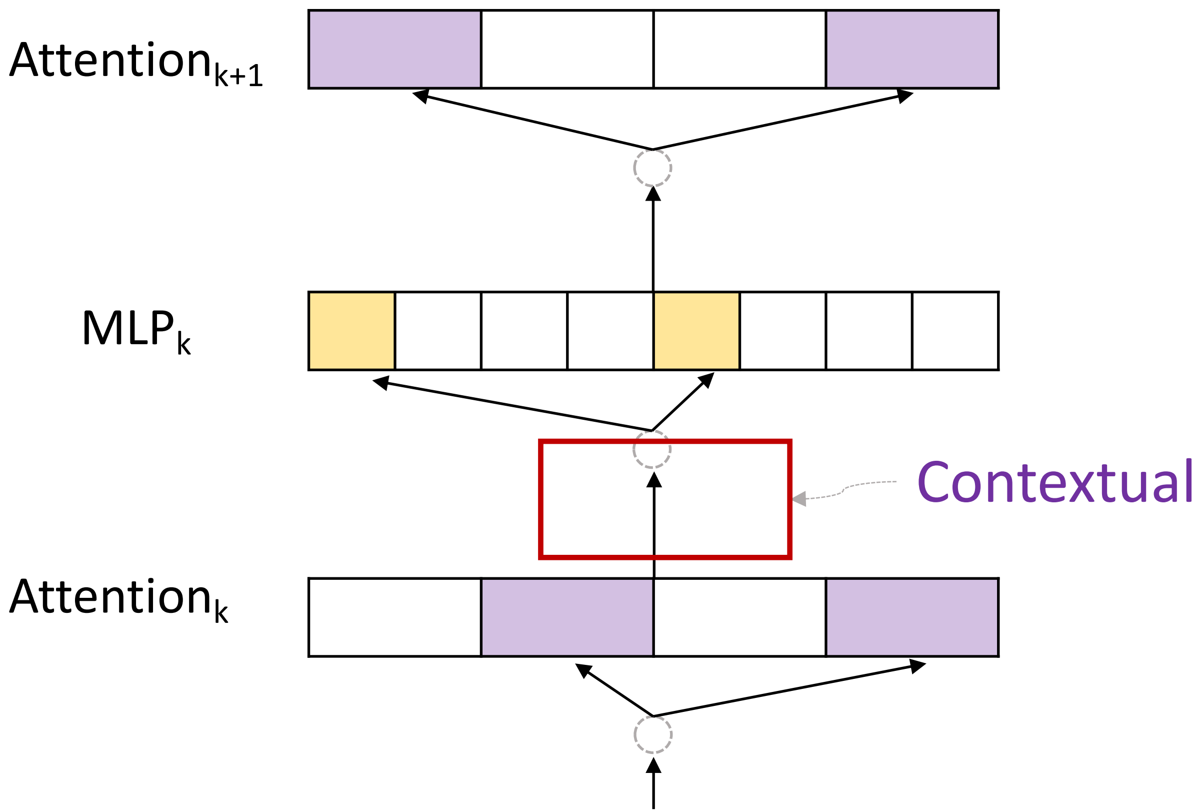
Contextual Sparsity: Existence



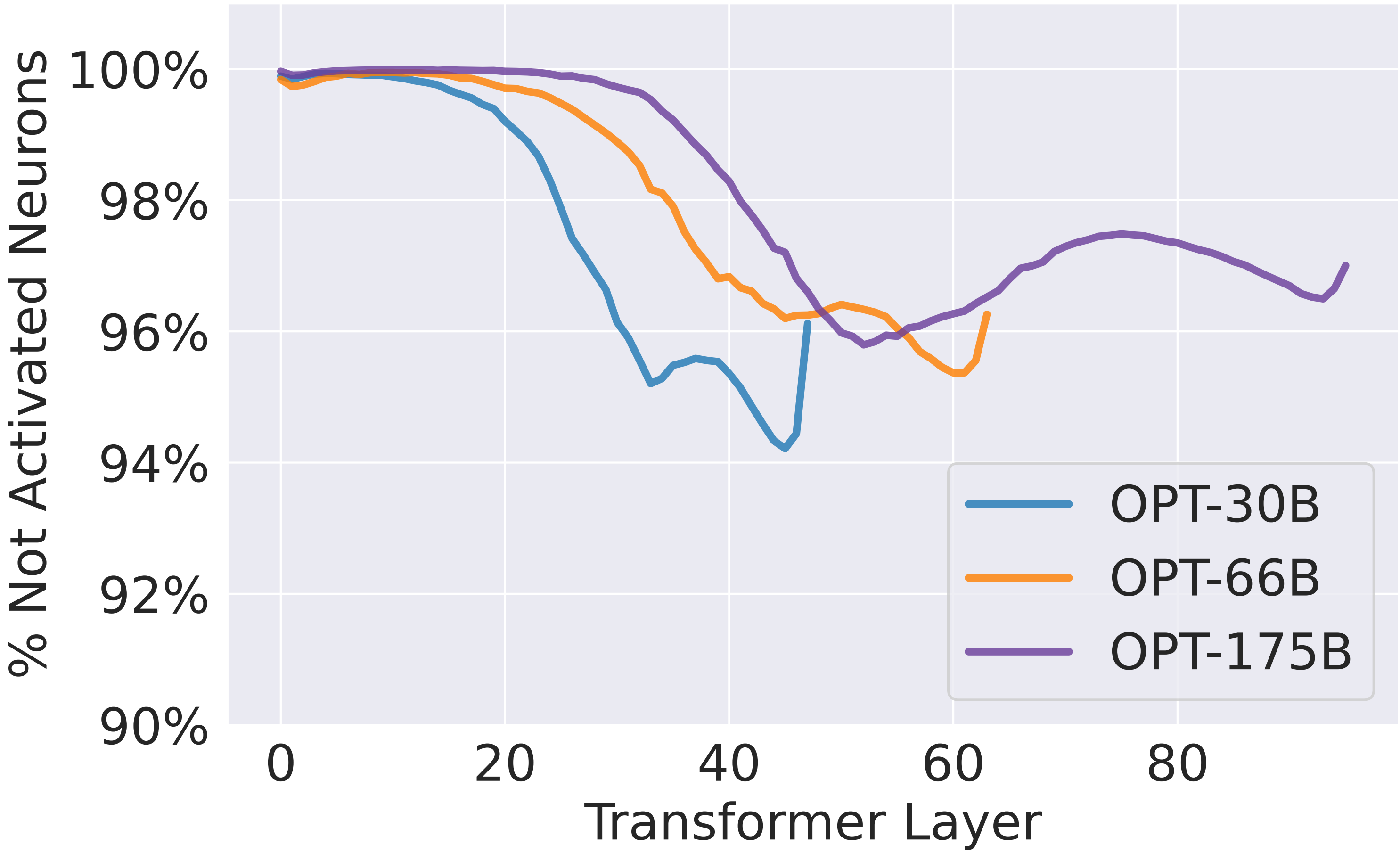


Contextual Sparsity: Existence



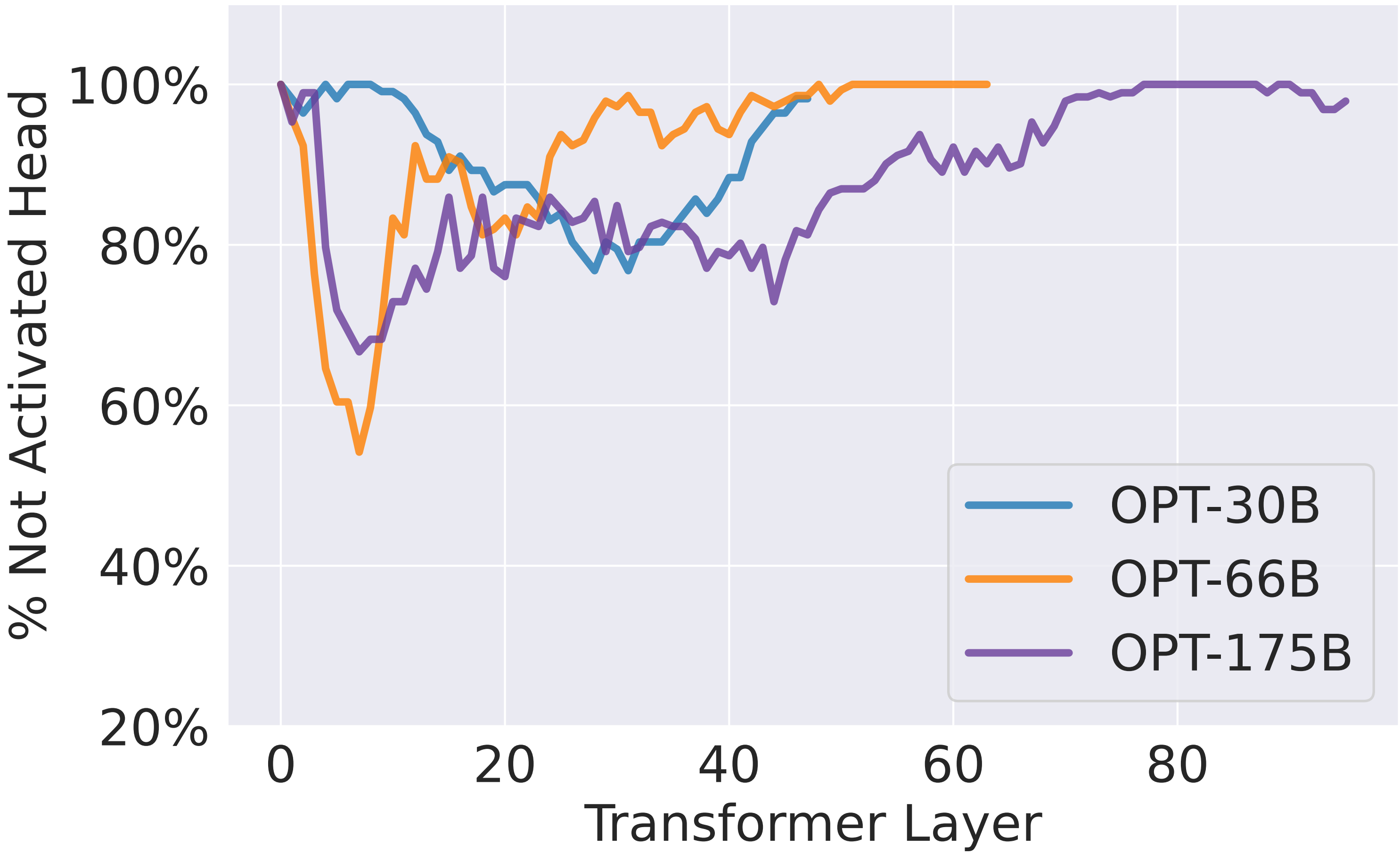


Contextual Sparsity Exists in MLPs

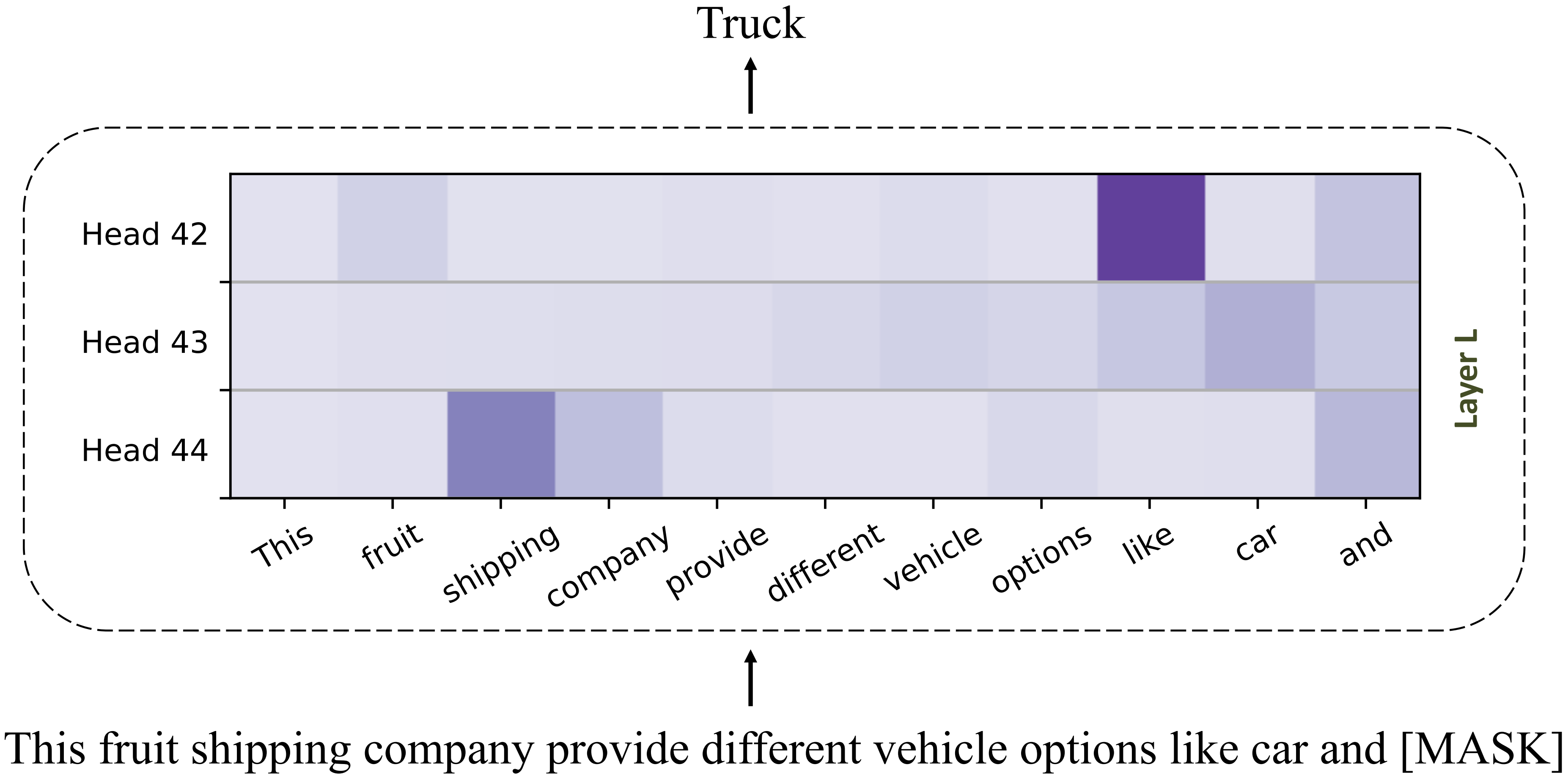


- Due to activation functions, e.g., ReLU, GeLU
- Similar observation in (Li et al.)

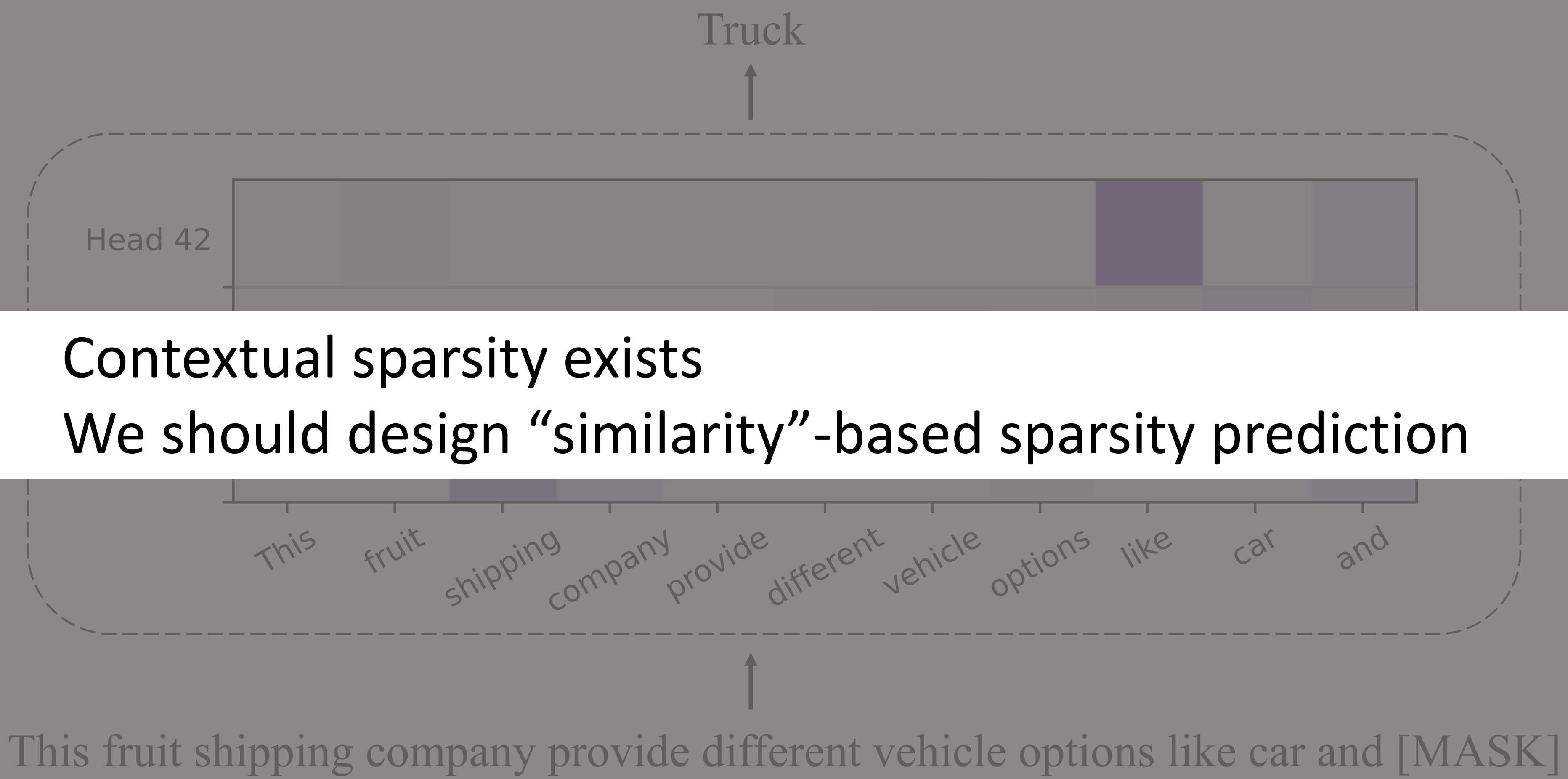
Contextual Sparsity Exists in Attention



Contextual Sparsity Exists in Attention

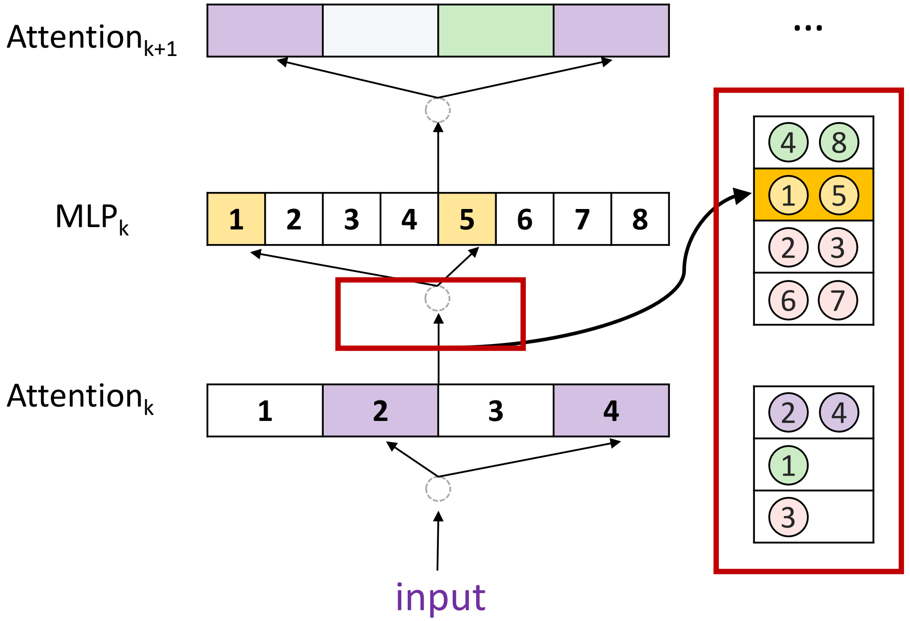


Contextual Sparsity Exists in Attention



- Contextual sparsity exists
- We should design “similarity”-based sparsity prediction

Contextual Sparsity: Prediction



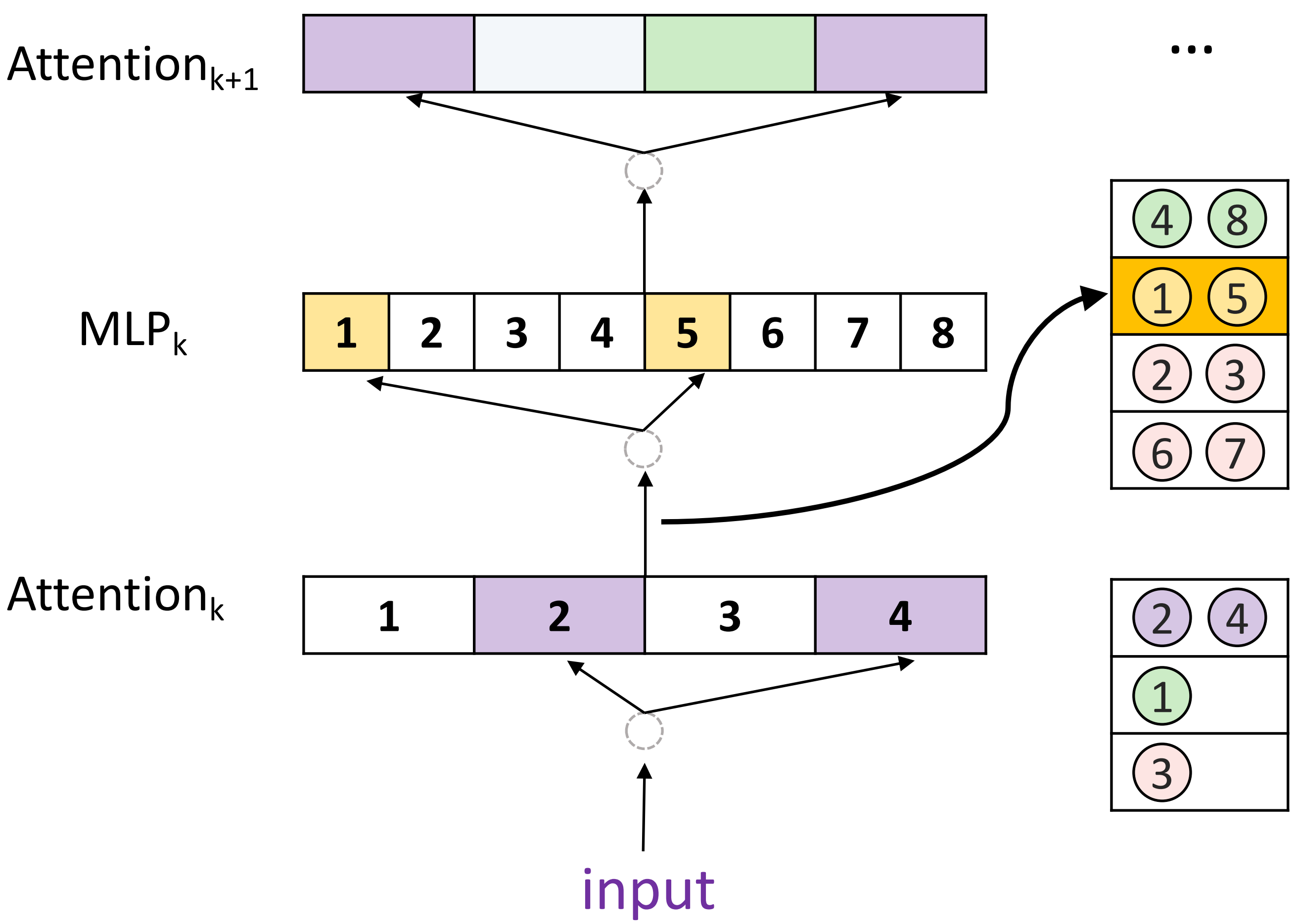
Challenge: how to predict high activation on-the-fly without computing the full attention or MLP?

Key idea: design a "similarity"-based prediction

- formulate the prediction problem as near-neighbor search (NNS).
- Data – neurons or attention heads
- Query – input at each layer

NNS algorithms can make prediction based on the similarity between input & parameters.

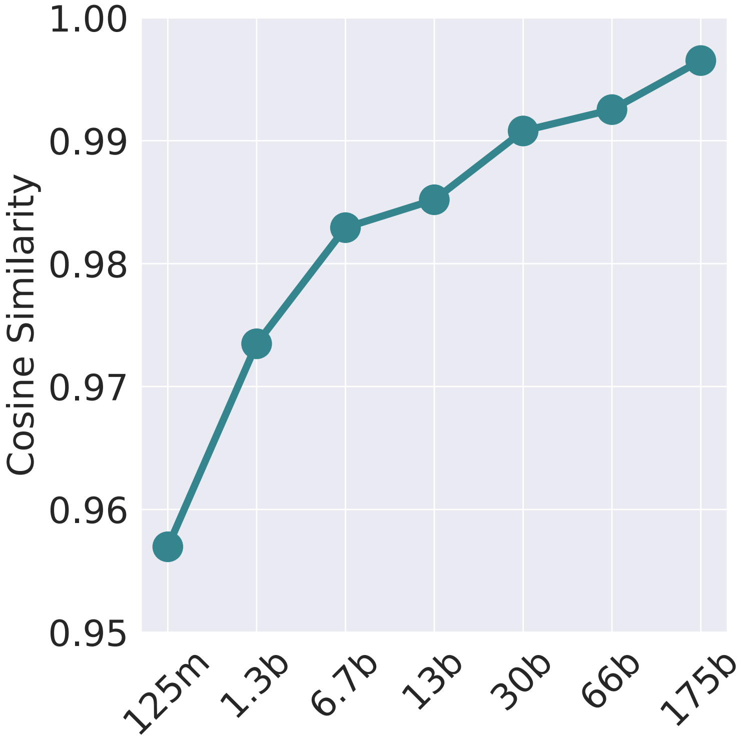
Contextual Sparsity: Efficiency

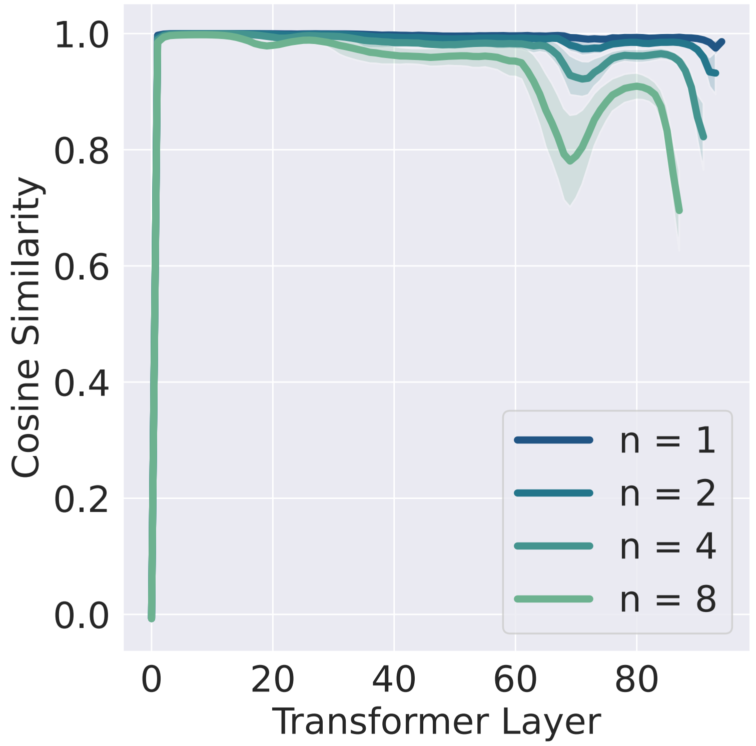


NNS overhead and SpMM

- it performs at each layer
- hash table is not efficient on GPU
- Sparse matmul complicates the implementation

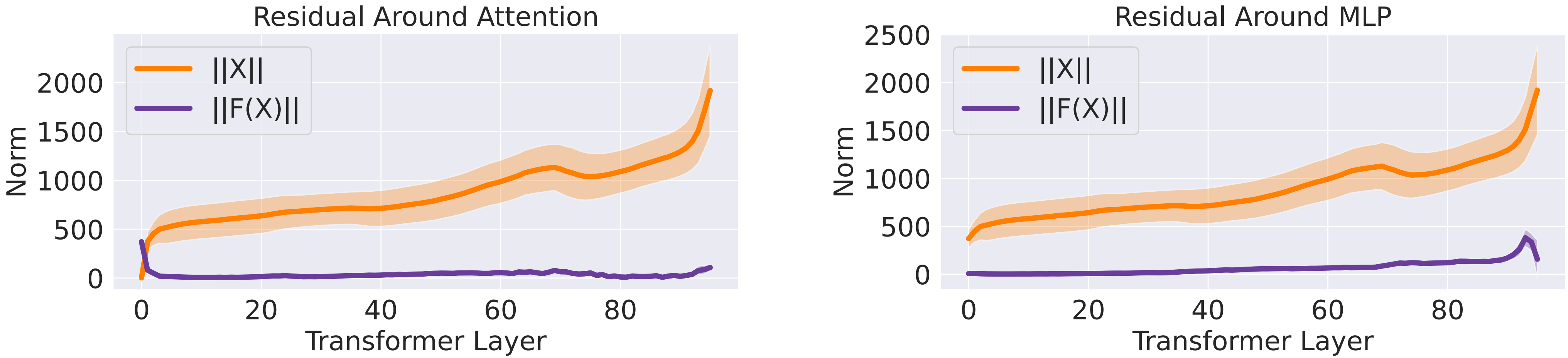
Key Insight: Slowly Changing Embeddings across Layers





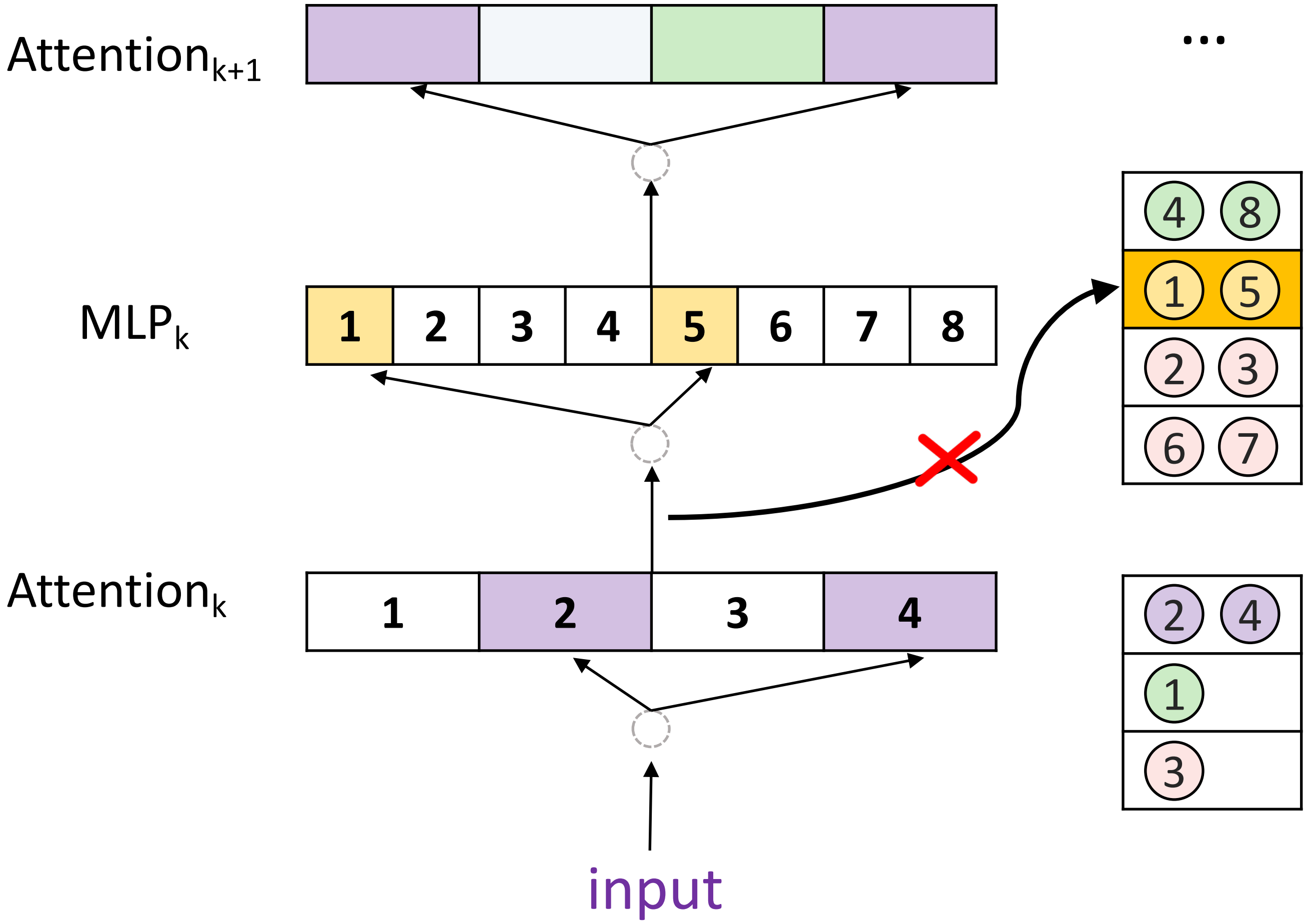
Cosine similarity between representations at consecutive layers is very high.

Key Insight: Slowly Changing Embeddings across Layers

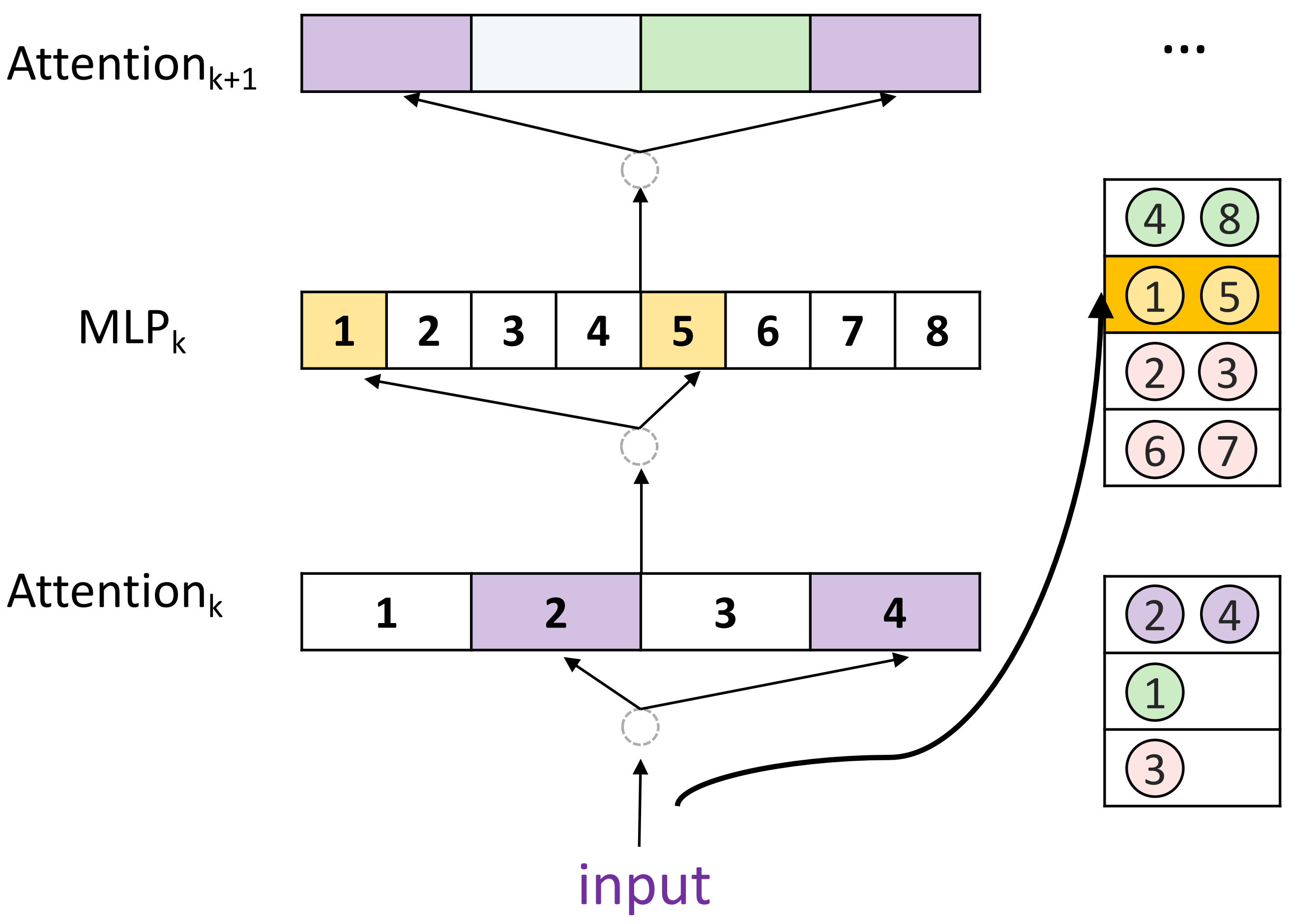


For the residual connection X' = X +F (X), X's norm dominates.

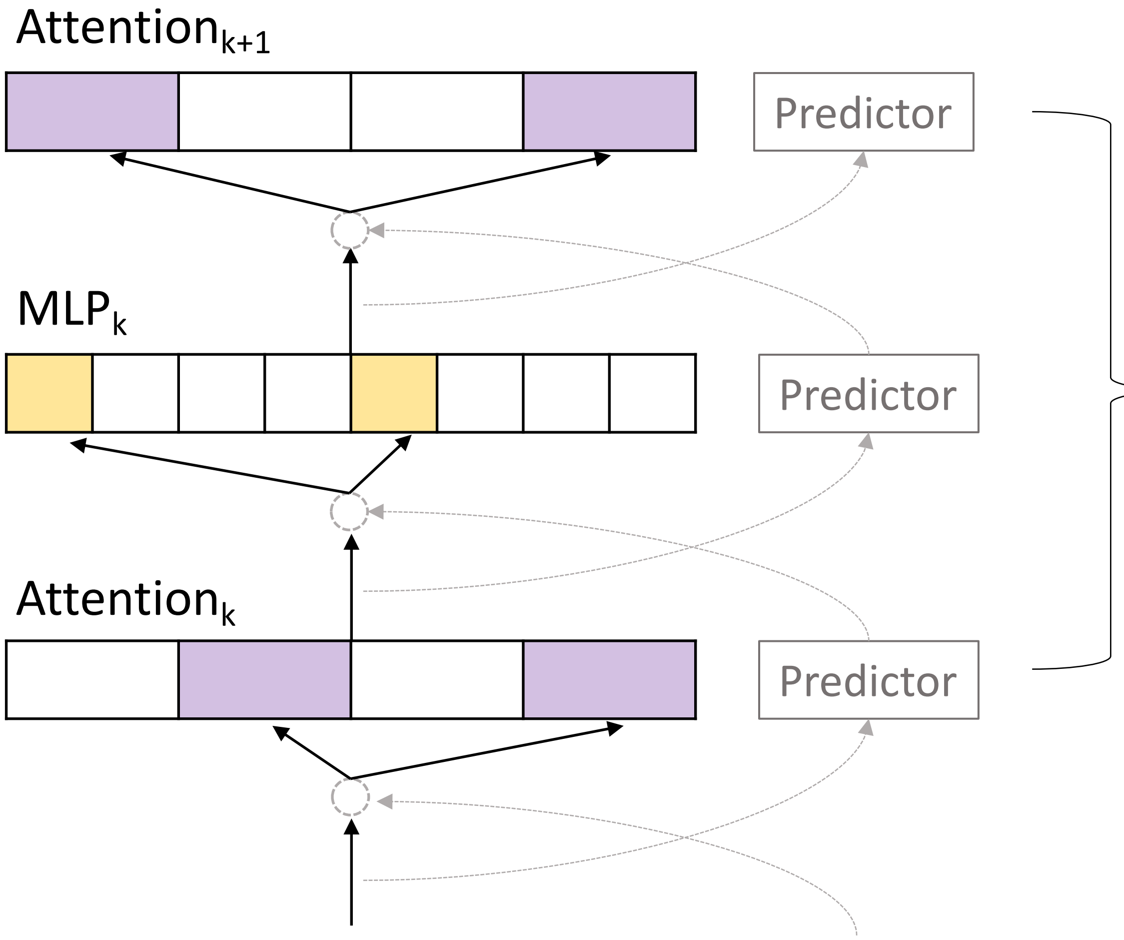
Predict contextual sparsity n-layers ahead



Reduce overhead with Asynchronous Execution

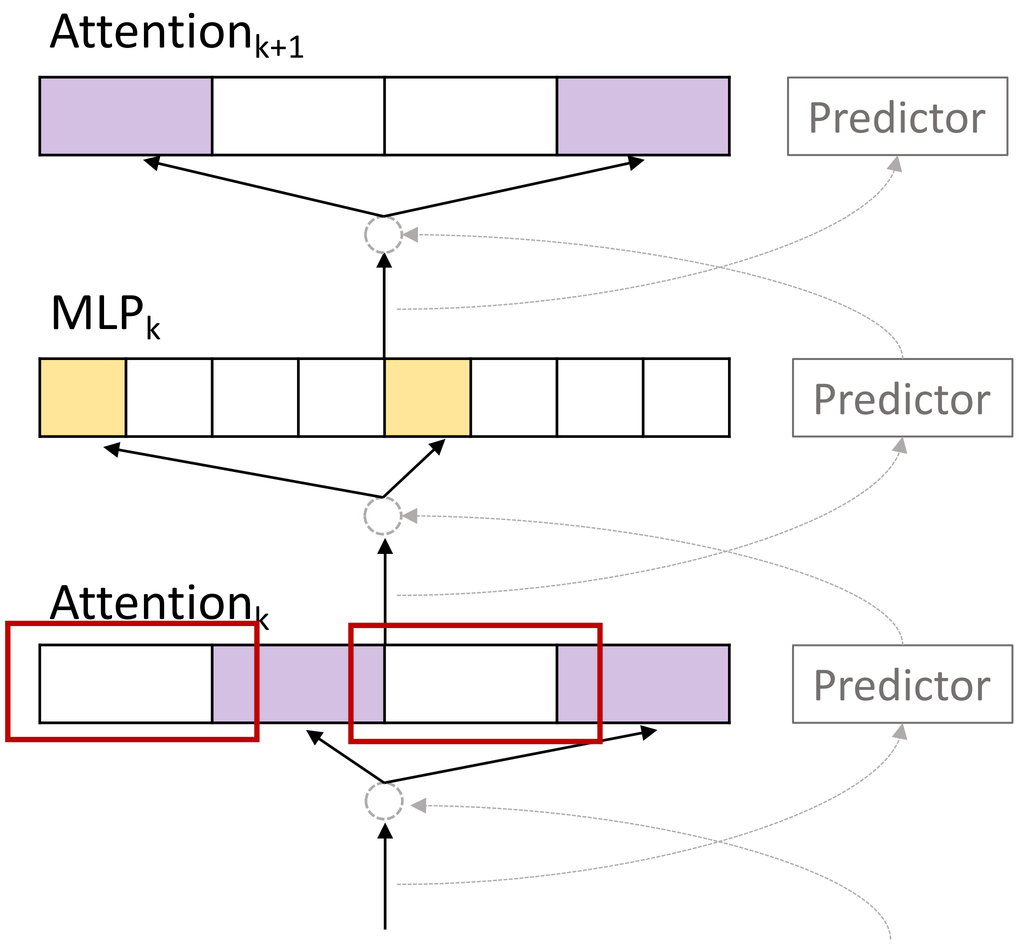


Reduce Overhead with GEMM-based Predictor



Low-cost small MLP predictors

Hardware-efficient Implementation



Kernel fusion:

- SpMM, indexing + multiplication
- Triton

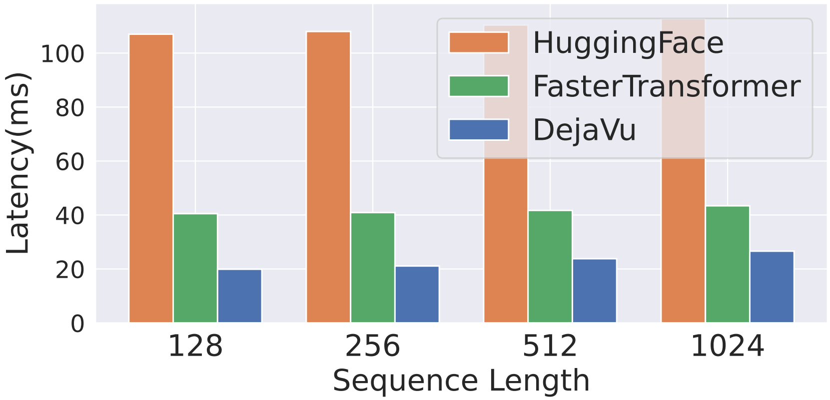
Memory coalescing:

- Store Atten out projection and MLP2 in column major

Missing KVCache for a past token:

- latency is bounded by weight loading
- Fill in missing ones when that head is loaded by a future token (compute is free)

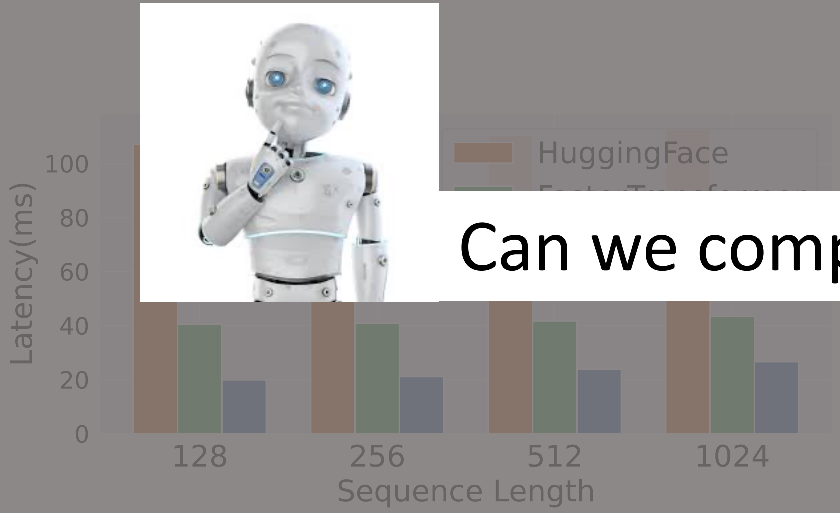
Deja Vu: 2X FasterTransformer and 6X HuggingFace



	COPA	OpenBookQA	Winogrande	Lambada
OPT-175B	0.86	0.446	0.726	0.758
Deja Vu-OPT-175B	0.85	0.45	0.726	0.753
OPT-175B + W4A16	0.85	0.44	0.714	0.757
Deja Vu-OPT-175B + W4A16	**0.86**	**0.452**	**0.726**	**0.754**

- demonstrates best performance with batch size=1, ReLU, 175B model
- maintains accuracy even combined with quantization.
- achieves speed up with larger batch size, more activation functions, and smaller models.

Deja Vu: 2X FasterTransformer and 6X HuggingFace



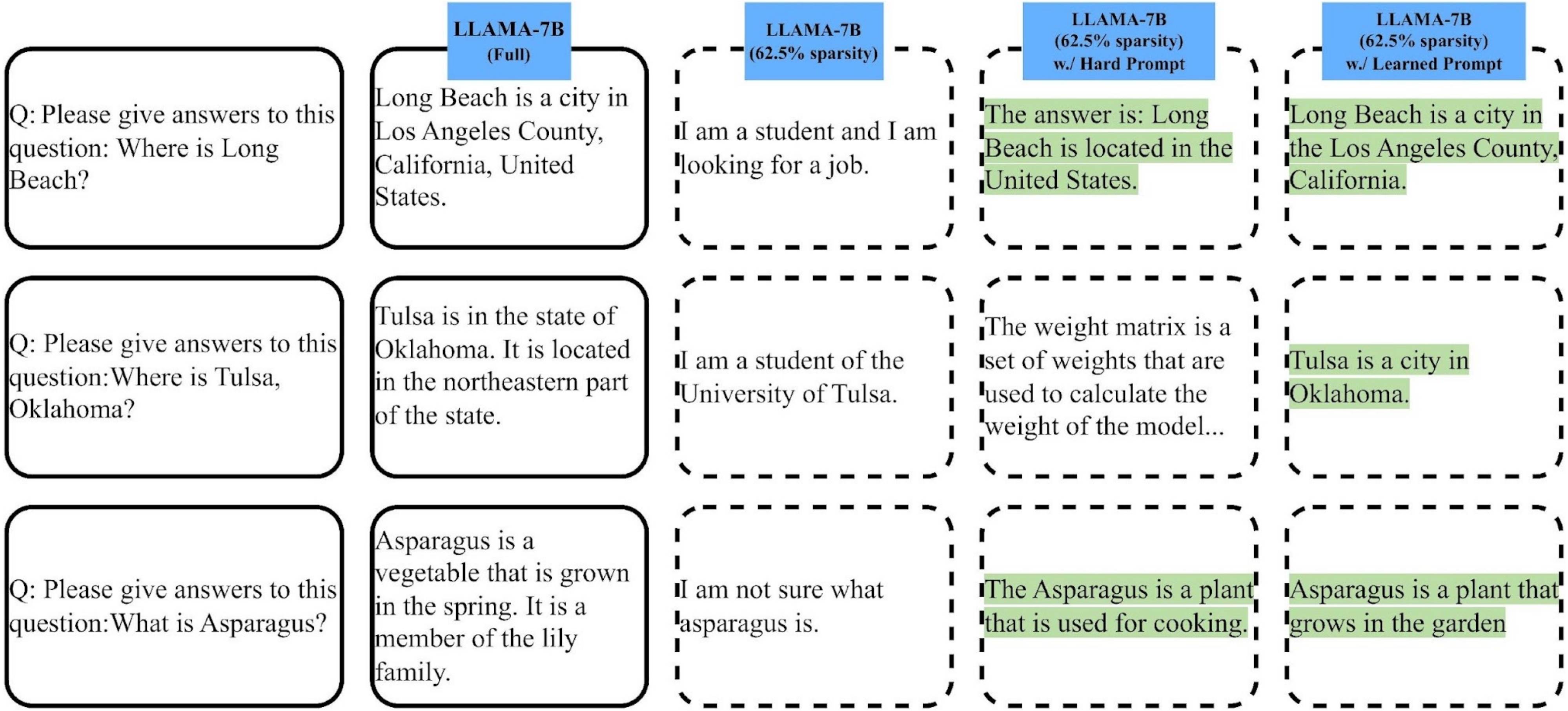
Can we compress more?

	COPA	OpenBookQA	Winogrande	Lambada
[illegible]B	0.86	0.446	0.726	0.758
[illegible]-175B	0.85	0.45	0.726	0.753
OPT-175B + W4A16	0.85	0.44	0.714	0.757
Deja Vu-OPT-175B + W4A16	**0.86**	**0.452**	**0.726**	**0.754**

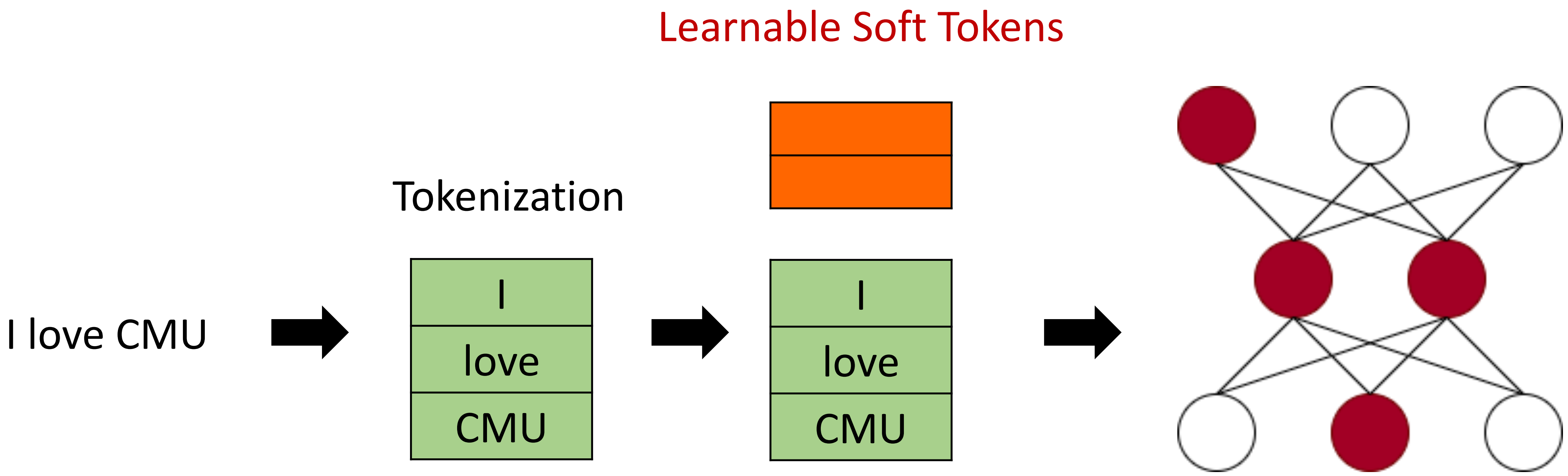
- demonstrates best performance with batch size=1, ReLU, 175B model
- maintains accuracy even combined with quantization.
- achieves speed up with larger batch size, more activation functions, and smaller models.

Observation: Prompt Helps SparseGPT pruned models

Prompt: Please carefully examine the weight matrix within the model, as it may contain errors. It is crucial to verify its accuracy and make any necessary adjustments to ensure optimal performance



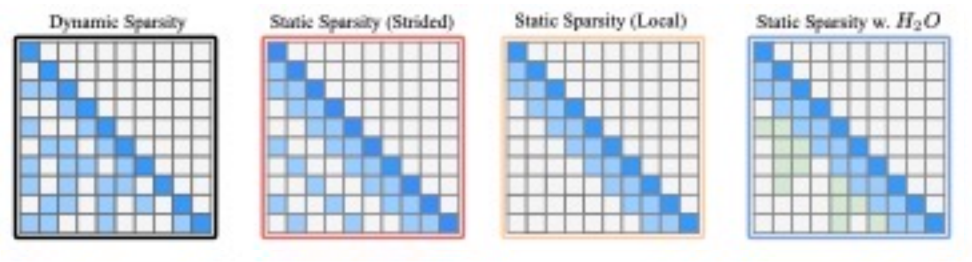
Prompt Learning Strategy for Compressed LLMs



Soft Prompt are better and transferable across tasks and different compression techniques.

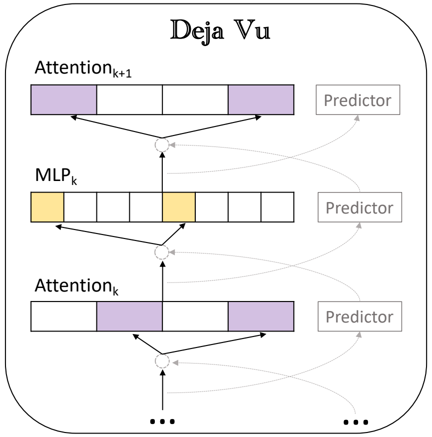
LLMs are Powerful , but Very Expensive to Deploy

H_2O (*NeurIPS'23*)



29x, 29x, 3x higher throughput, 1.9x lower latency than DeepSpeed Zero-Inference, HuggingFace Accelerate, and FlexGen with Heavy-Hitter Sparsity.

Deja Vu (*ICML'23*)



2x lower latency than FasterTransformer and 6x than HuggingFace on 8xA100 with contextual sparsity.

StreamingLLM (*new* 🔥)

Model 4 million tokens… 22x faster than sliding window recomputation with Attention Sink.

Compress, Then Prompt (*new* 🔥)

8x extreme model compression (Sparse+Quantize) with Prompt Recovery.

LLMs are Powerful , but Very Expensive to Deploy

H_2O *(NeurIPS'23)*

29x, 29x, 3x higher
than DeepSpeed Z
Accelerate, and Fle

Deja Vu *(ICML'23)*

ver latency than
rTransformer and
an HuggingFace
A100 with
xtual sparsity.

QUUUUIZ Time

Is it possible to load LLM < 100 times when generating / decoding 100 tokens from the same distribution? If so, how?

StreamingLLM *(new 🔥)*

Model 4 million tokens… 22x faster than sliding window recomputation with Attention Sink.

Compress, Then Prompt *(new 🔥)*

8x extreme model compression (Sparse+Quantize) with Prompt Recovery.

Thanks You!

Q&A