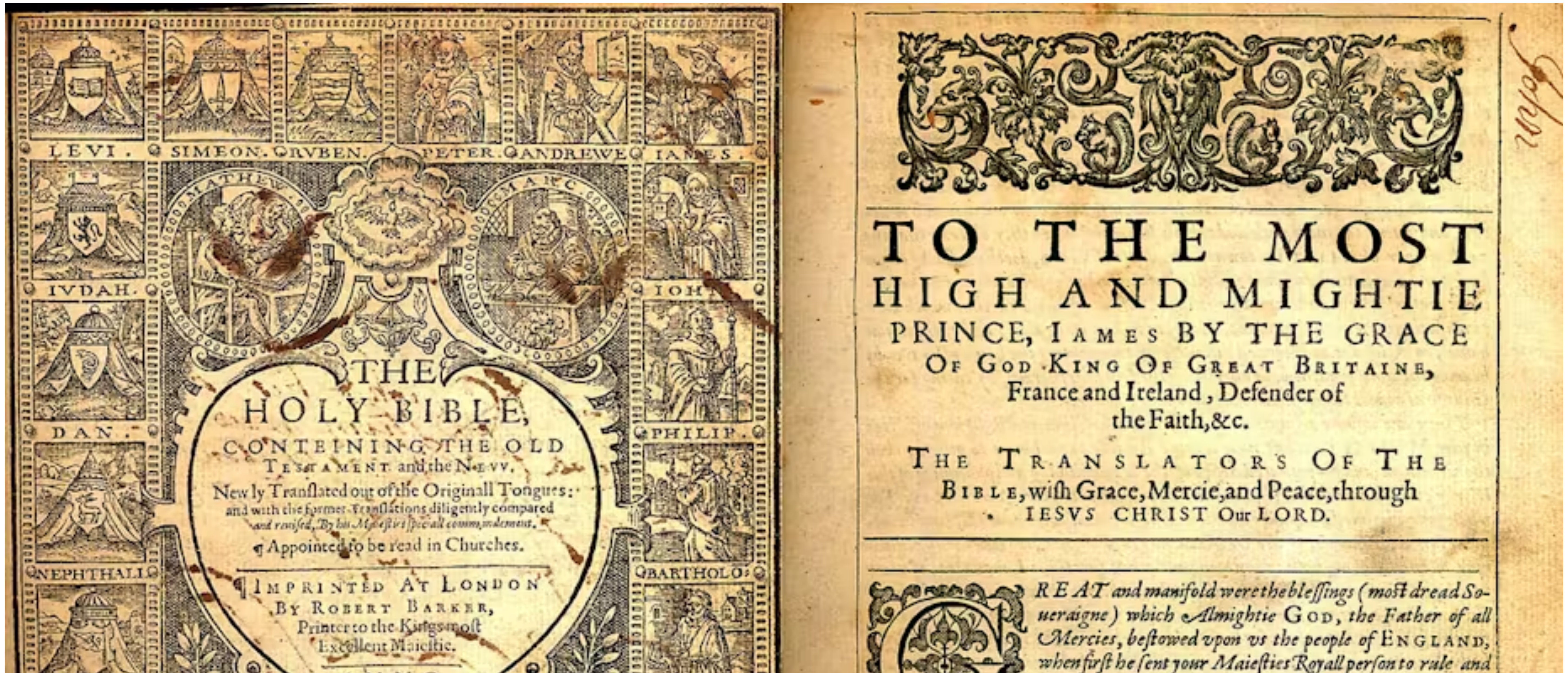
Long context LLMs

EECE695D: Efficient ML Systems

Spring 2025

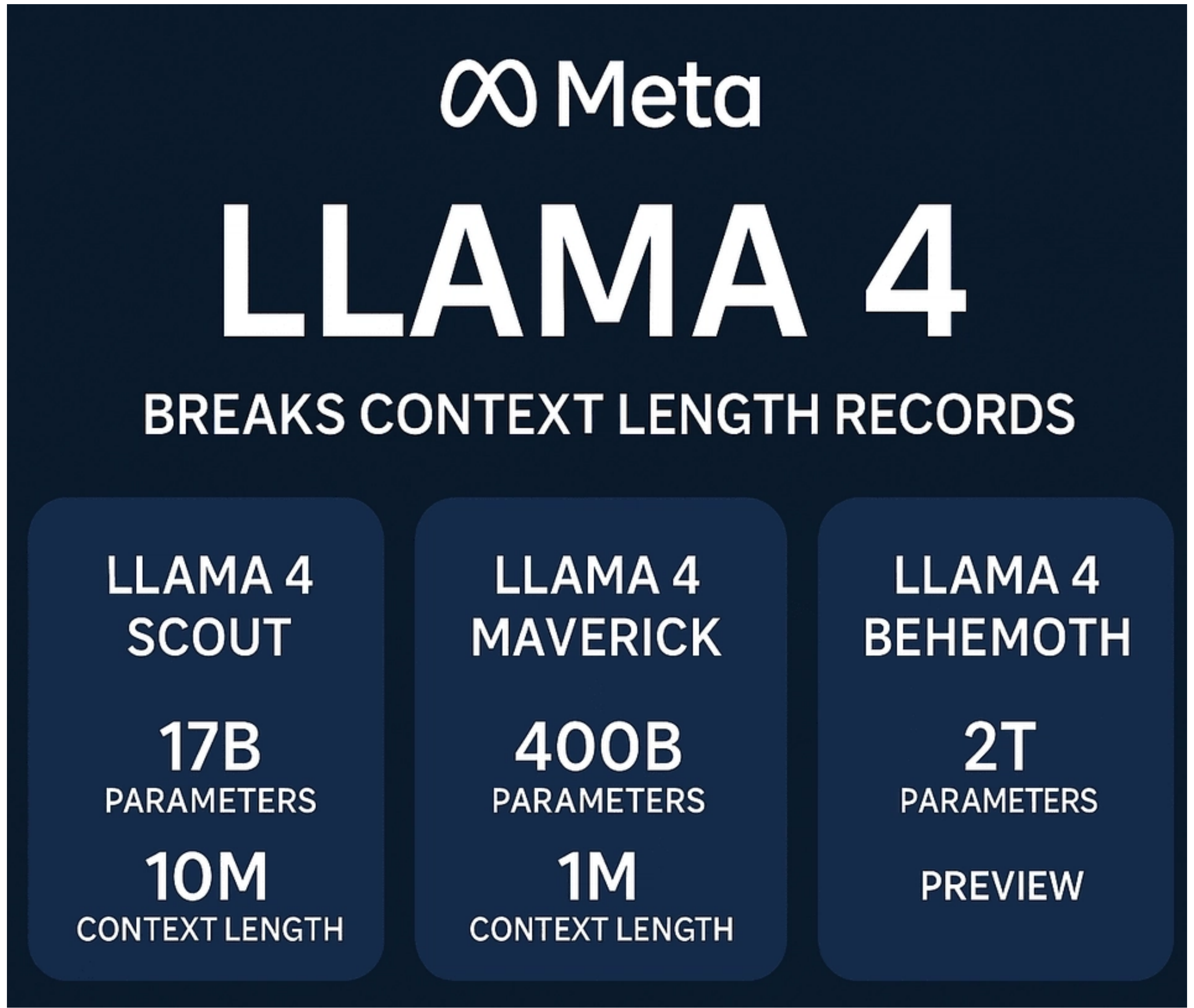
Handling long contexts

- Suppose that we give LLM a new book and ask questions about it
- **Question.** How much context length would we need?
    - e.g., King James Bible has **783,137 words ≈ 1M tokens**



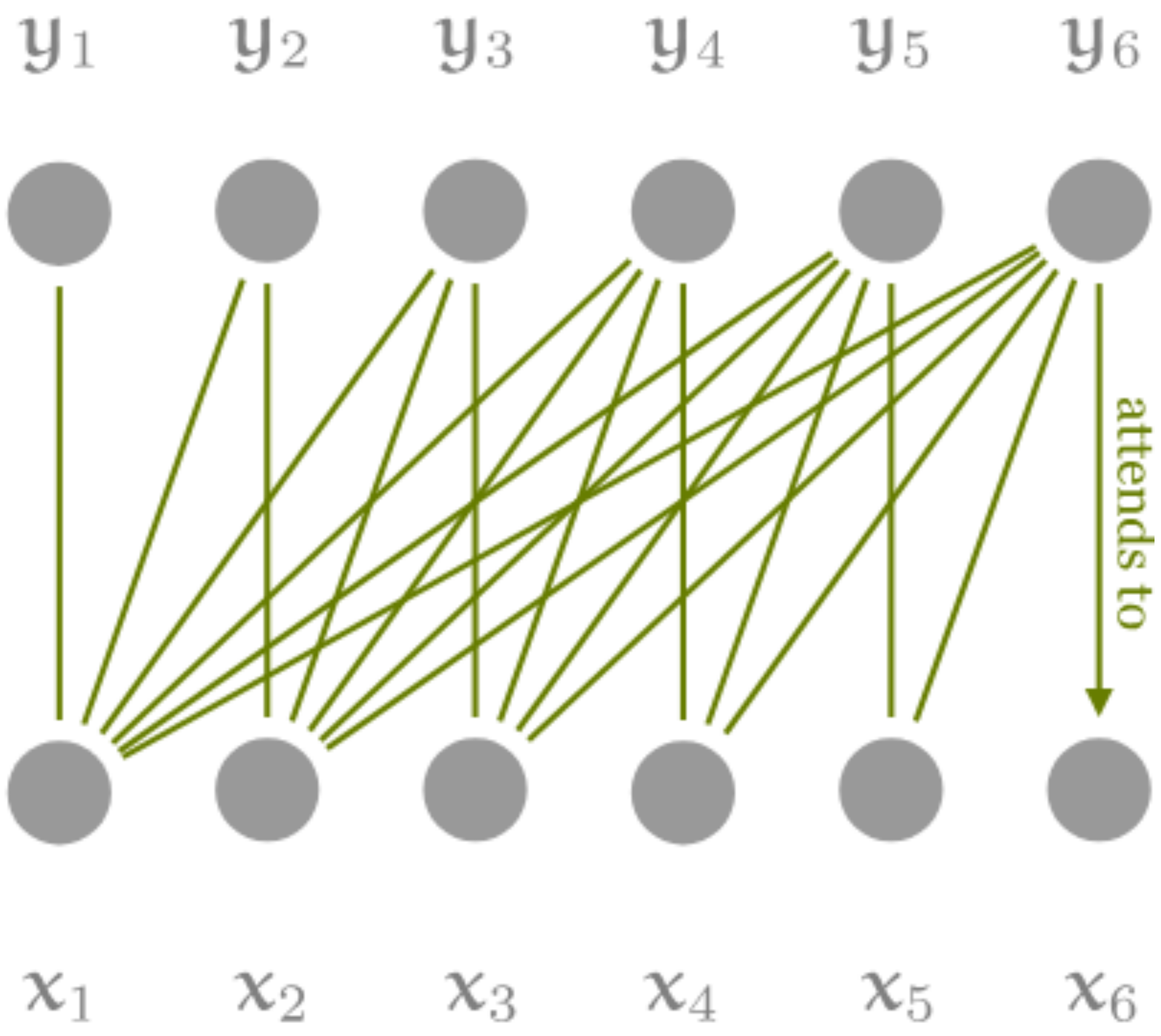
Handling long contexts

- Thus, everybody loves to have LLMs that can handle **long context**
  - Multimodal input, e.g., video
  - Inference-time scaling
- However, LLMs typically had limited context lengths, until very recently…
  - LLaMA 1: 2k
  - LLaMA 2: 4k
  - LLaMA 3: 8k
  - LLaMA 4: ***10M***



Handling long contexts

- **Why?** Long context is expensive, computationally
    - Computational cost of self-attention layers grow quadratically
    - Much memory I/O at generation phase



Handling long contexts

- **Solution.**
    - FlashAttention
    - Train short, then extend
    - Compress the KV cache

FlashAttention

Computing self-attention

- A simple technique to reduce **memory I/O** for long context
- Recall the self-attention operation

$$\mathbf{O} = \sigma(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$$

- This is done by:
  - Compute $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$
  - Materialize:
    - $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N \times N}$
    - $\mathbf{P} = \sigma(\mathbf{S}) \in \mathbb{R}^{N \times N}$
    - $\mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d}$

Computing self-attention

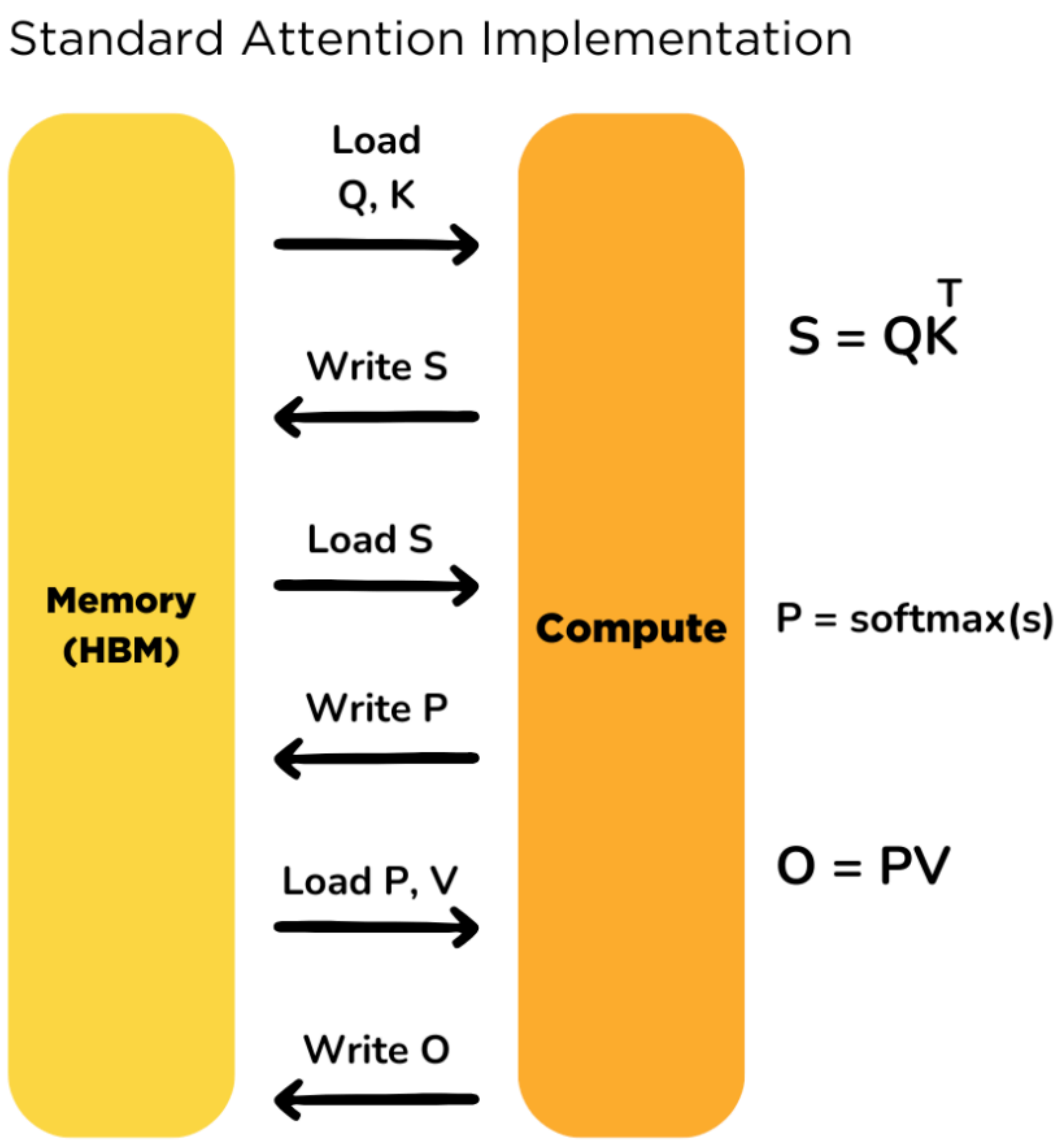
$$\mathbf{S} = \mathbf{Q}\mathbf{K}^{\top} \in \mathbb{R}^{N \times N}$$

$$\mathbf{P} = \sigma(\mathbf{S}) \in \mathbb{R}^{N \times N}$$

$$\mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d}$$

- Naïvely, requires **much HBM I/O**
  - Division (tokens) and fusion (ops) desired
    - Difficult due to softmax

$$[\sigma(x)]_j = \frac{\exp(x_j)}{\sum_i \exp(x_i)}$$



https://huggingface.co/docs/text-generation-inference/en/conceptual/flash_attention

Concretely...

- Consider processing j-th input token

$$\mathbf{o}_j = \sum_{i=1}^{N} \mathbf{v}_i \cdot [\sigma(\mathbf{q}_j^\top \mathbf{k}_:)]_i$$

- **Standard Algo.** Two loops:
  - For $i \in \{1, \ldots, N\}$, accumulate the denominator

$$\mathbf{z}_i = \mathbf{z}_{i-1} + \exp(\mathbf{q}_j^\top \mathbf{k}_i)$$

  - For $i \in \{1, \ldots, N\}$, compute the weighted sum:

$$\mathbf{o}_i = \mathbf{o}_{i-1} + \mathbf{v}_i \cdot \frac{\exp(\mathbf{q}_j^\top \mathbf{k}_i)}{\mathbf{z}_N}$$

- **If N is large.** Requires re-loading Q,K and re-computing the dot products

Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS 2022

FlashAttention

- Does this in a **single loop**
- **Idea.** Consider a surrogate sequence

$$\mathbf{o}'_i = \sum_{j=1}^{i} \mathbf{v}_j \cdot \frac{\exp(\mathbf{q}^\top \mathbf{k}_j)}{\mathbf{z}_i}$$

which is a partial sum normalized by another partial sum.

  - Satisfies two properties:
    - $\mathbf{o}'_N = \mathbf{o}_N$
    - Follows the recurrence relation

$$\mathbf{o}'_i = \frac{\mathbf{o}'_{i-1} \cdot \mathbf{z}_{i-1} + \mathbf{v}_i \cdot \exp(\mathbf{q}^\top \mathbf{k}_i)}{\mathbf{z}_i}$$

Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS 2022

FlashAttention

- **Algo.** For $i = \{1, \ldots, N\}$, compute:
    - $e_i = \exp(\mathbf{q}^\top \mathbf{k}_i)$
    - $\mathbf{z}_i = \mathbf{z}_{i-1} + e_i$
    - $\mathbf{o}'_i = \dfrac{\mathbf{o}'_{i-1} \cdot \mathbf{z}_{i-1} + \mathbf{v}_i \cdot e_i}{\mathbf{z}_i}$ $\Leftarrow$ Can do kernel fusion!

- **Advantages**
    - Constant memory on SRAM
        - independent of N
    - Loading $\mathbf{k}_i, \mathbf{v}_i$ only once

Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS 2022

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0×)
GPT-2 small - FlashAttention	1k	18.2	**2.7 days (1.7×)**
GPT-2 small - FlashAttention	2k	17.6	3.0 days (1.6×)
GPT-2 small - FlashAttention	4k	**17.5**	3.6 days (1.3×)

Further readings

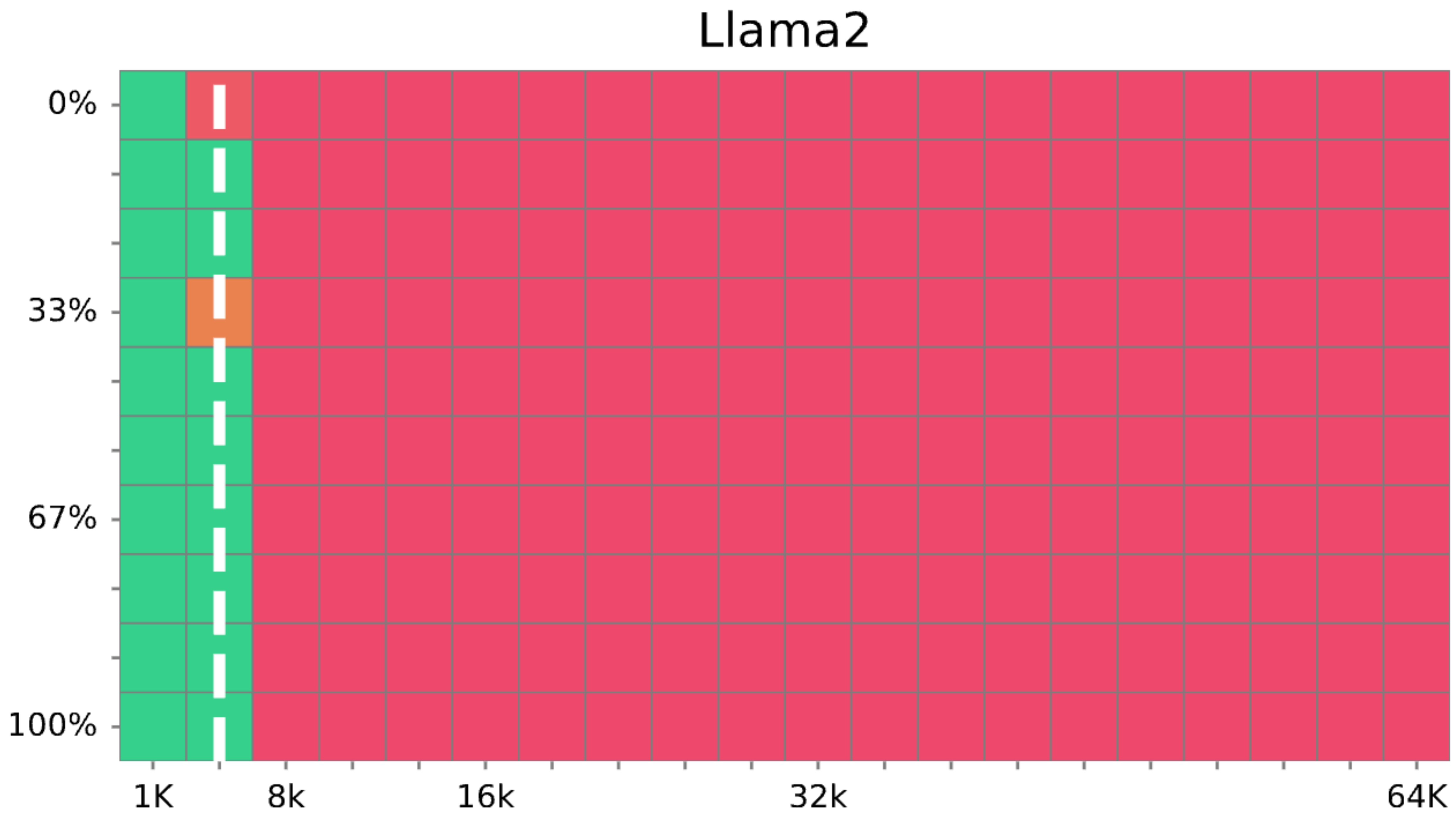
- Self-Attention does not need $O(N^2)$ memory: https://arxiv.org/abs/2112.05682
- FlashAttention 1: https://arxiv.org/abs/2205.14135
  - How to handle backward
  - How to handle "safe softmax"
  - Block-sparse
- FlashAttention 2: https://arxiv.org/abs/2307.08691
  - Parallelism and work partitioning
  - Reducing non-matmul FLOPs

Dao et al., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," NeurIPS 2022

Extending context length

Length generalization

- **Idea.** Train a model with short context window, and use it for long context.
- **Problem.** Without any tricks, does not generalize
  - Example. Needle in a haystack; retrieving a word at specific position (white dotted line: context length of training)



Lu et al., "A Controlled Study on Long Context Extension and Generalization in LLMs," arXiv 2024

Length generalization

- Popular solutions involve altering positional embeddings
- **RoPE.** A relative positional embedding for transformers
  - Rotates query / key by a certain degree, based on positions
  - Example. For the j-th input token

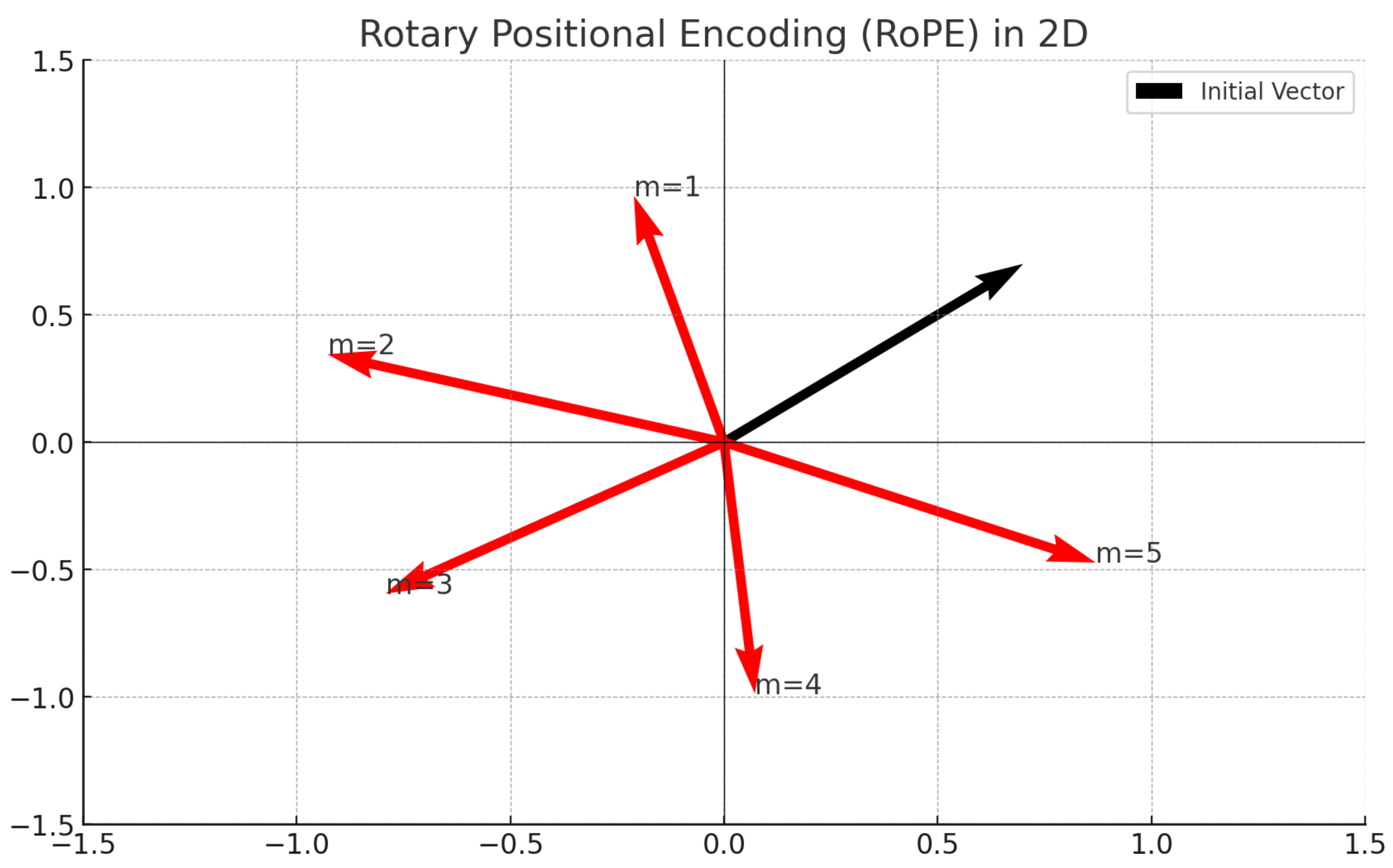
$$f_q(\mathbf{x}_j, j) = \mathbf{R}_{\Theta,m}\mathbf{W}_q\mathbf{x}_j, \qquad f_k(\mathbf{x}_j, j) = \mathbf{R}_{\Theta,m}\mathbf{W}_k\mathbf{x}_j$$

$$\mathbf{R}_{\Theta,j} = \begin{bmatrix} \cos j\theta_1 & -\sin j\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin j\theta_1 & \cos j\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos j\theta_2 & -\sin j\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin j\theta_2 & \cos j\theta_2 & \cdots & 0 & 0 \\ \cdots & \cdots & & & \cdots & \cdots & \cdots \end{bmatrix}$$

Su et al., "RoFormer: Enhanced Transformer with Rotary Position Embedding," arXiv 2021

Length generalization

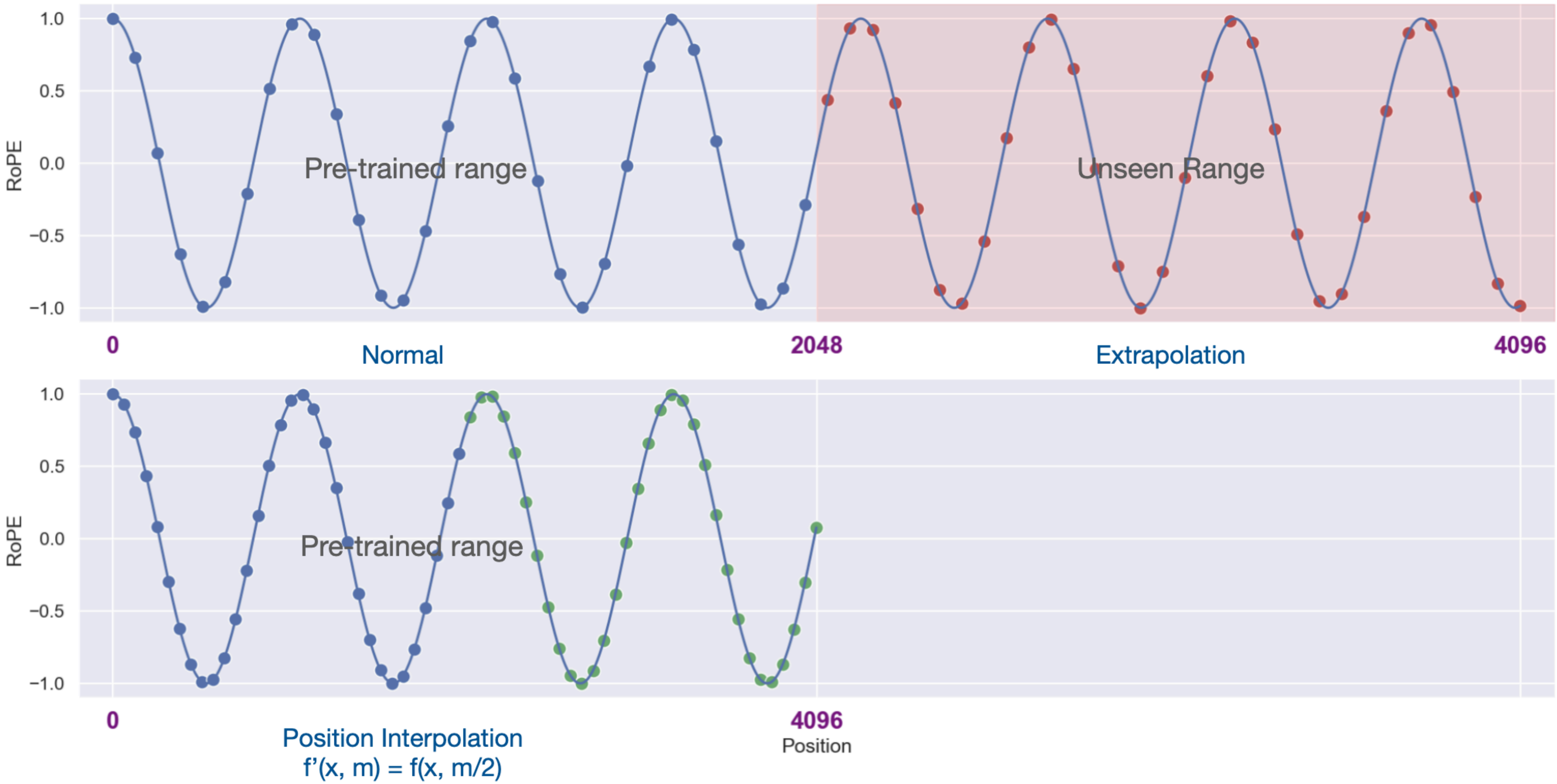
- RoPE is useful because it preserves the dot product of query-key after any shift in the token locations
  - Same if tokens lie at (2,4) or (10002,10004)
- Apply rotation for two-dimension-chunks



Su et al., "RoFormer: Enhanced Transformer with Rotary Position Embedding," arXiv 2021

Position interpolation

- **Idea.** Reduce the frequency by $1/K$, to increase the context length K-fold



Chen et al., "Extending Context Window of Large Language Models via Positional Interpolation," arXiv 2023

"Dynamic NTK" interpolation

- Proposed by a redditor "emozilla"

- **Idea.** Apply different scaling to different frequencies
  - Large θ: Scale down less
  - Small θ: Scale down more
  - Intuition. High-frequency θ are sensitive to relative positions
    - These are thus precious; better keep them intact

https://www.reddit.com/r/LocalLLaMA/comments/14mrgpr/dynamically_scaled_rope_further_increases/

Further readings

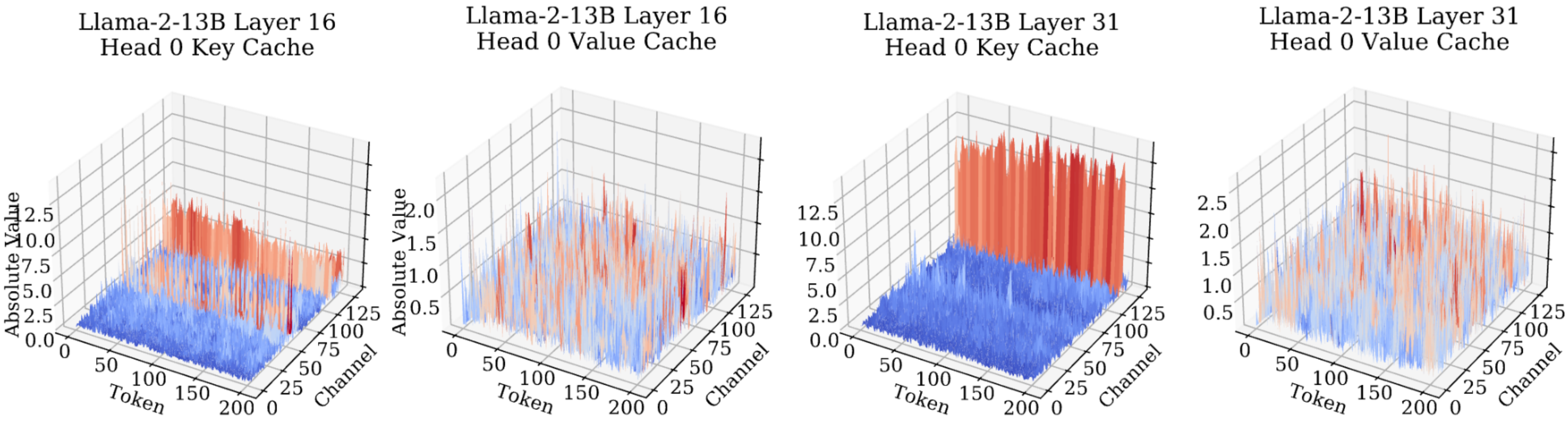
- YaRN: https://openreview.net/forum?id=wHBfxhZu1u
    - Additional temperature scaling
- A controlled comparison: https://arxiv.org/abs/2409.12181
    - Careful comparison, where NTK-RoPE is the winner

- Attention patterns (Long LoRA): https://arxiv.org/abs/2309.12307
    - Fine tuning

KV cache compression

KV cache compression

- **Idea.** Instead of compressing weights & activations, compress the KV cache
  - More about memory I/O than computation
- **Quantization** (e.g., FlexGen)
- Again, there are outliers that we should worry about — in keys



Liu et al., "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache," arXiv 2024

KV cache compression

- Similar tricks can be used:
  - Hadamard rotation
  - Weight migration — do not migrate to weight, but scale up the queries!

$$\mathbf{Z} = (\mathbf{Q\Lambda}) \cdot (\mathbf{K\Lambda}^{-1})^\top, \qquad \mathbf{\Lambda} = \mathrm{diag}(\lambda)$$

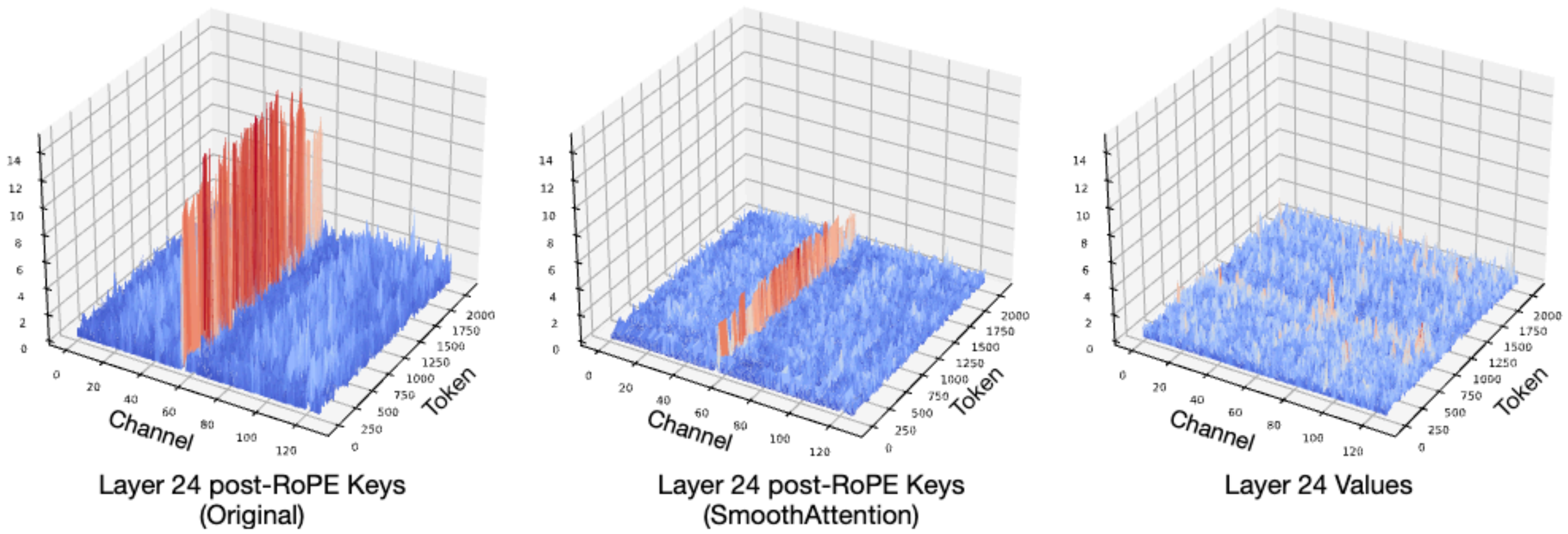
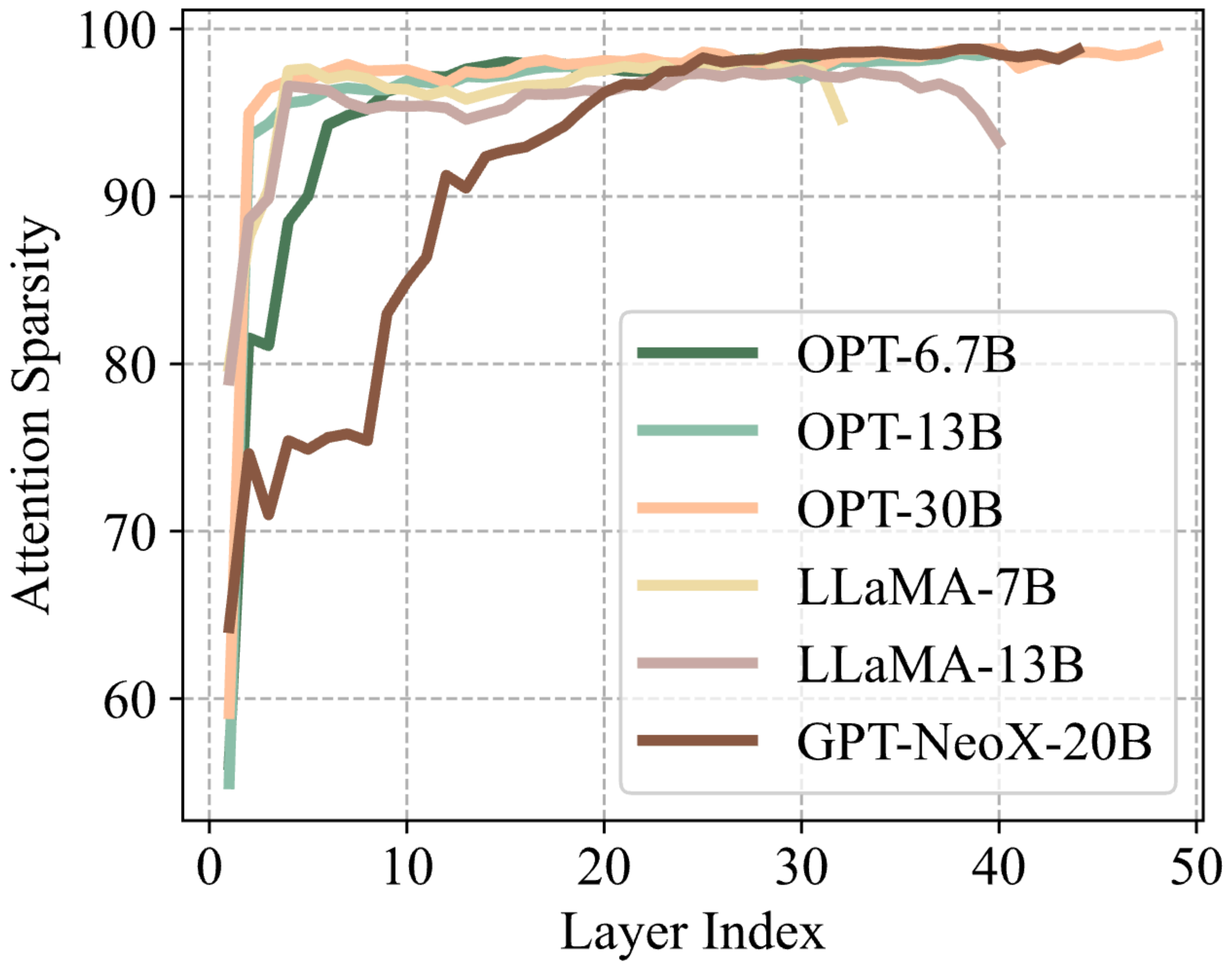


Fig. 7: **SmoothAttention effectively smooths the outliers in Keys. Values doesn't suffer from outliers.**

Lin et al., "QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving," arXiv 2024

KV cache compression

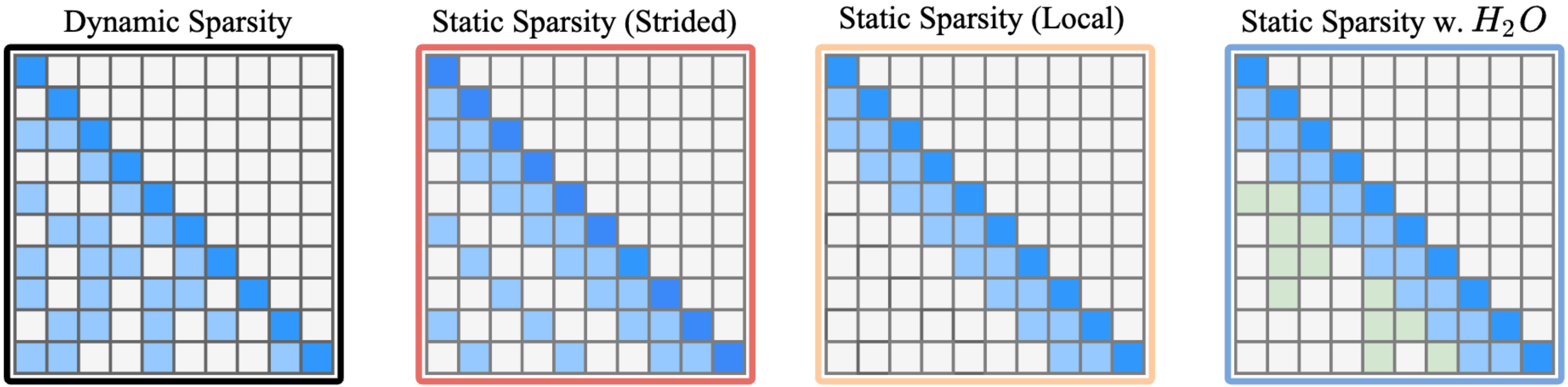
- **Sparsity** (e.g., H2O)
- Turns out that only a fraction of tokens matter for the future generation
    - (Right) Sparsity of the softmax, where we consider zero when less than 1% of maximum
- Thus, load only a fixed number of important tokens



Zhang et al., "H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models," NeurIPS 2023

KV cache compression

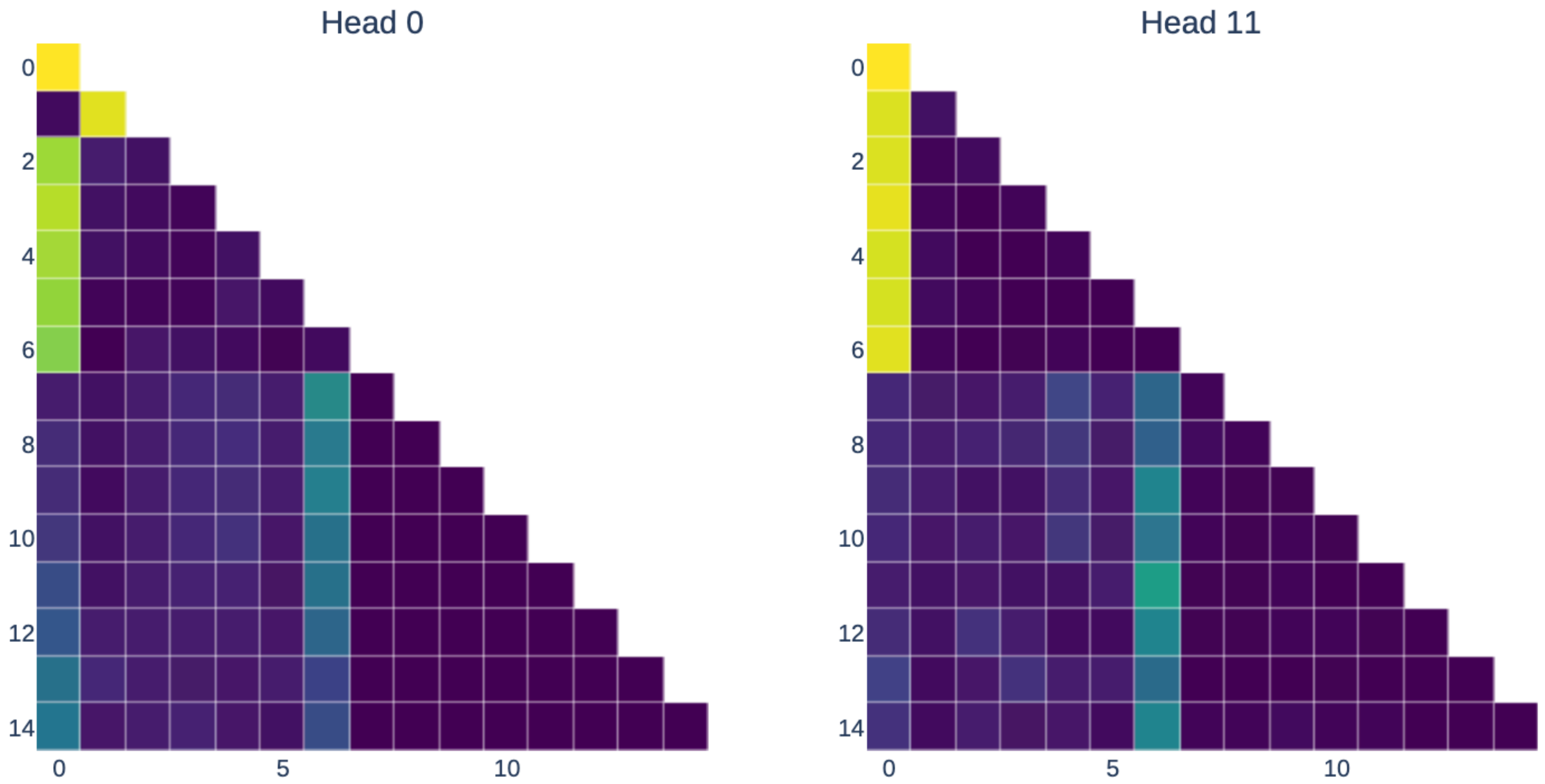
- **H2O.** Recent token + important
  - Keep track of the weighted sum of attention scores
    - Lightweight heuristic to approximate these



Zhang et al., "H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models," NeurIPS 2023

KV cache compression

- **Sink-based.** Attention sinks (meaningless but takes up all attention) have much impact on the output
  - Diagnosable with small ℓ_2 norm of the value token
  - Keep bottom-k tokens



Devoto et al., "A Simple and Effective L2 Norm-Based Strategy for KV Cache Compression," EMNLP 2024

Wrapping up

- We did not cover **"efficient transformers"** literature
  - Reformer: https://arxiv.org/abs/2001.04451
    - Slightly outdated, but still provides a nice use of locality-sensitive hashing (LSH)

That's it for today 🙌