10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Long Context in LLMs

Matt Gormley & Henry Chai

Lecture 19

Nov. 6, 2024

Reminders

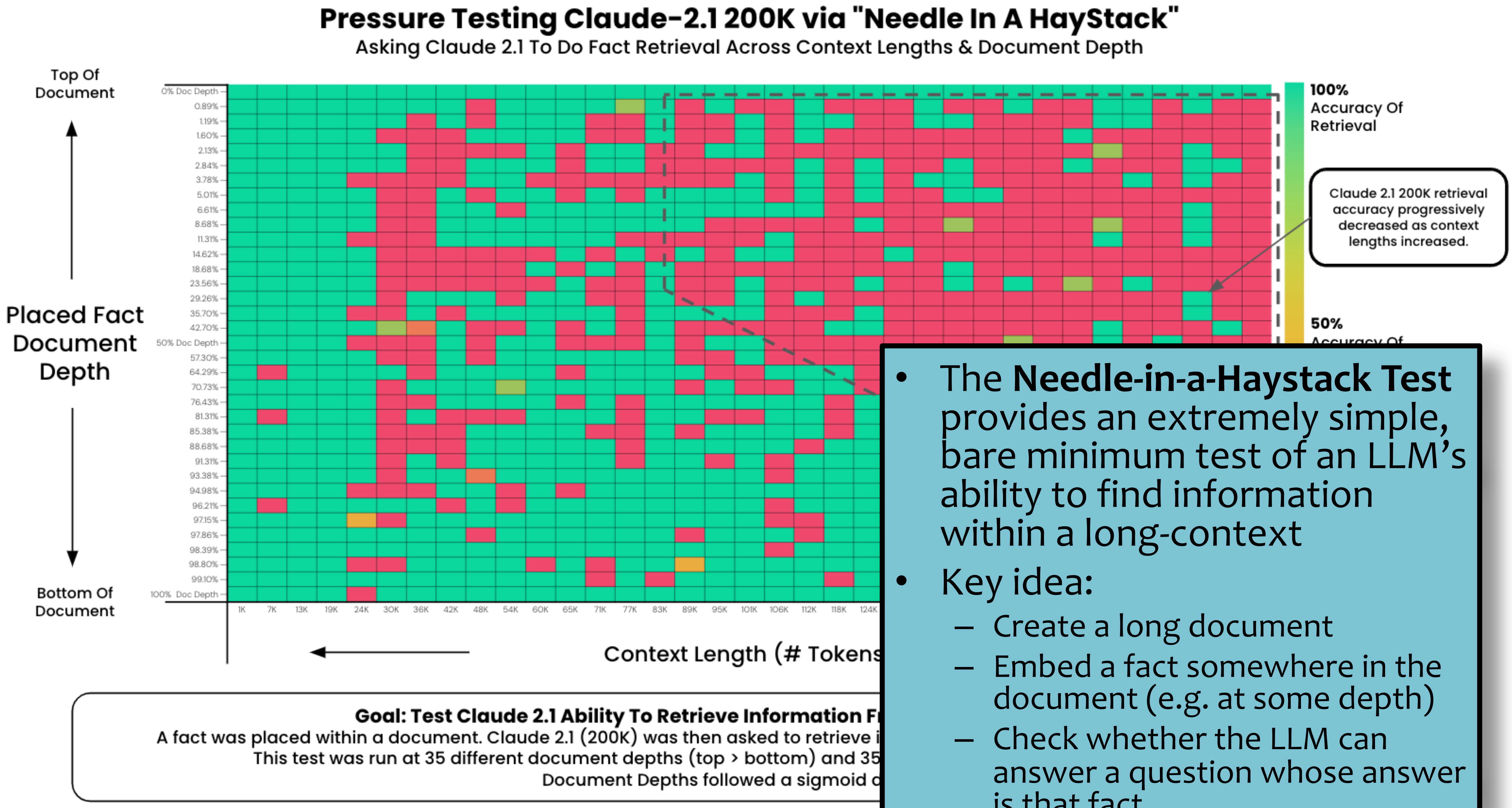
- Exam
  - Date: In-class, Monday, Nov 11
  - Time: 75 minutes, taking up the whole class time
  - Covered Material: Lectures 1 – 15 (same as Quiz 1 – Quiz 4)
  - You may bring one sheet of notes (front and back)
  - Format of questions: Unlike the Quiz questions, which were all multiple choice, Exam questions will include open-ended questions as well
  - Check Piazza for seat assignment

LONG-CONTEXT LLMS

Context Length of Transformer LMs

Comparison of some recent **large language models** (LLMs) on their **context size**, i.e. how many tokens they can accept

Model	Creators	Year	Model Size	Context Size
GPT-2	OpenAI	2019	1.5 billion	1024
GPT-3	OpenAI	2020	175 billion	2048
PaLM	Google	2022	540 billion	2048
LLaMA	Meta	2023	65 billion	2048
LLaMA-2	Meta	2023	70 billion	4096
Claude-2	Anthropic	2023	? (130 billion)	100k
Claude-2.1	Anthropic	2023	? (130 billion)	200k
GPT-4	OpenAI	2023	? (1.76 trillion)	8192
Mistral	Mistral AI	2023	7 billion	8192 (32k)
Mixtral	Mistral AI	2023	47 billion	8192 (128k)
Gemini (Ultra)	Google	2023	? (1.5 trillion)	32k
LWModel	academia!	2023	7 billion	1 million
Gemini-1.5	Google	2024	? (1.5 trillion)	1 million
GPT-4o	OpenAI	2024	?	128k
LLaMA-3	Meta	2024	405 billion	8192
LLaMA-3.1	Meta	2024	405 billion	128k
Claude-3.5	Anthropic	2024	?	200k

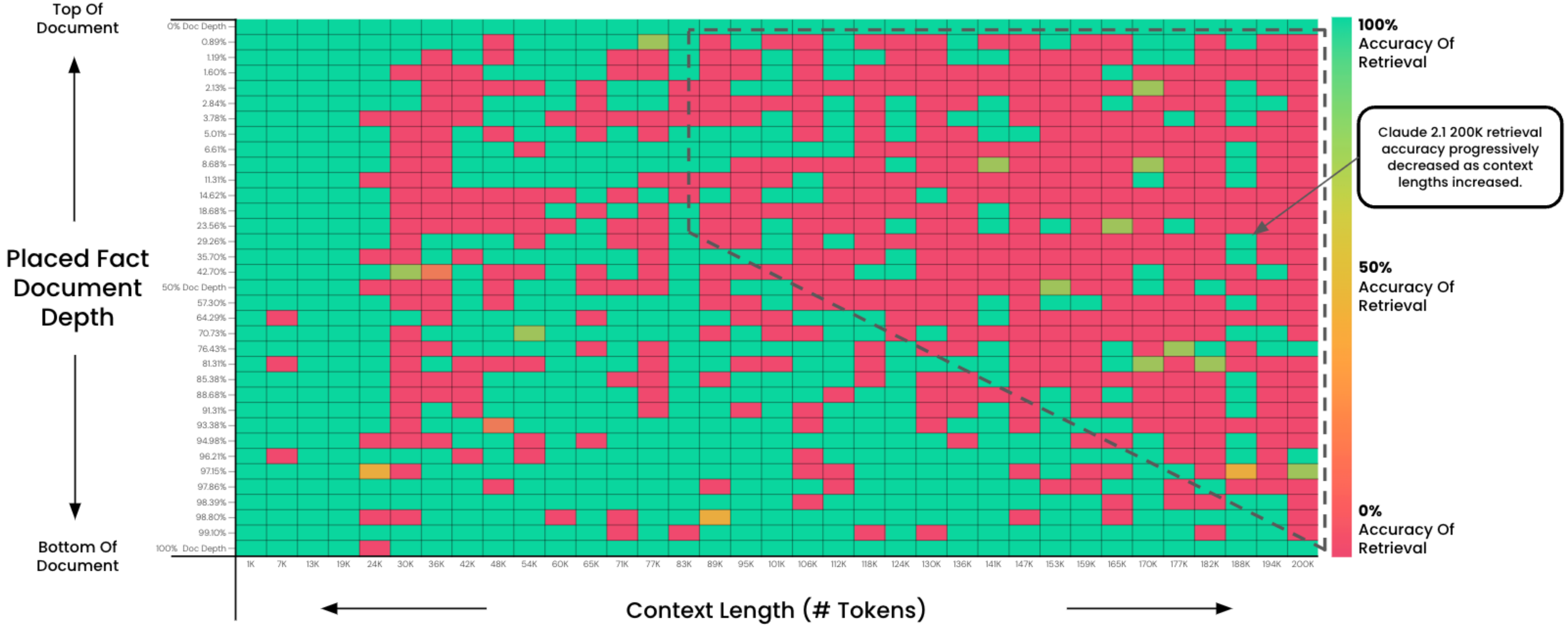


- The **Needle-in-a-Haystack Test** provides an extremely simple, bare minimum test of an LLM's ability to find information within a long-context
- Key idea:
  - Create a long document
  - Embed a fact somewhere in the document (e.g. at some depth)
  - Check whether the LLM can answer a question whose answer is that fact

Figure from https://github.com/gkamradt/LLMTest_NeedleInAHaystack

Pressure Testing Claude-2.1 200K via "Needle In A HayStack"

Asking Claude 2.1 To Do Fact Retrieval Across Context Lengths & Document Depth



Goal: Test Claude 2.1 Ability To Retrieve Information From Large Context Windows

A fact was placed within a document. Claude 2.1 (200K) was then asked to retrieve it. The output was evaluated (with GPT-4) for accuracy.
This test was run at 35 different document depths (top > bottom) and 35 different context lengths (1K >200K tokens).
Document Depths followed a sigmoid distribution

Figure from https://github.com/gkamradt/LLMTest_NeedleInAHaystack

Extending Short-Context Models

- There are two key ingredients for extending a short context model
  1. extensible positional embeddings
  2. careful selection of long data
- General recipe
  – Pre-train a short context model (block size = 4k) on 1 trillion tokens of text
  – Adjust the hyperparameters of the positional embeddings
  – Continue pre-training but increase the block size to support long-contexts (block size = 80k) on only 5 billion tokens of text

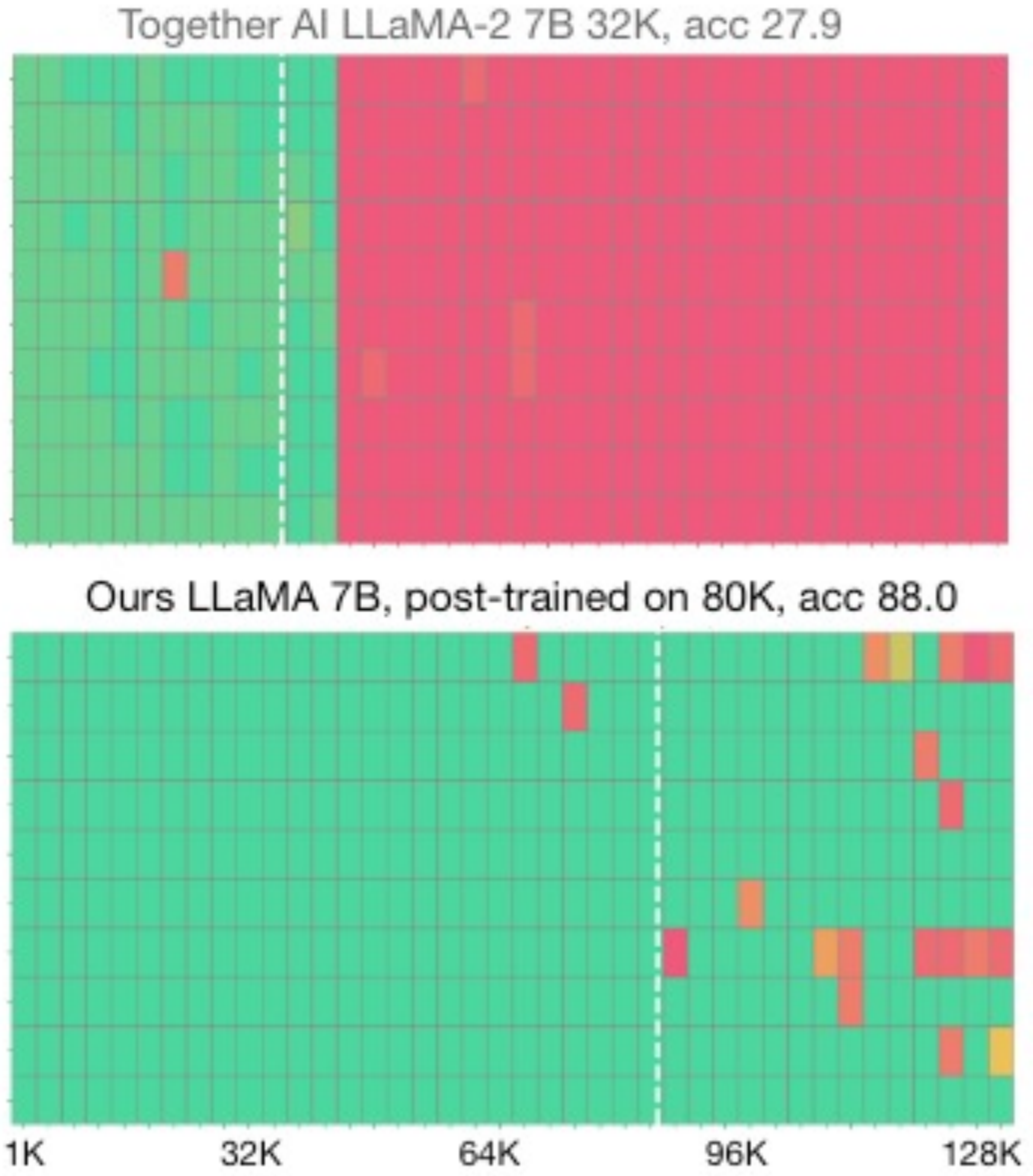


Figure from http://arxiv.org/abs/2402.10171

Fine-Tuning vs. In-Context Learning

- Why would we ever bother with fine-tuning if it's so inefficient?
- Because, even for very large LMs, fine-tuning often beats in-context learning
- In a fair comparison of fine-tuning (FT) and in-context learning (ICL), we find that FT outperforms ICL for most model sizes on RTE and MNLI

ICL \ FT	125M	350M	1.3B	2.7B	6.7B	13B	30B
125M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
350M	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
1.3B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
2.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
6.7B	−0.00	0.01	0.02	0.03	0.12	0.14	0.09
13B	−0.04	−0.02	−0.01	−0.00	0.09	0.11	0.05
30B	−0.11	−0.09	−0.08	−0.08	0.02	0.03	−0.02

(a) RTE

ICL \ FT	125M	350M	1.3B	2.7B	6.7B	13B	30B
125M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
350M	−0.00	0.00	0.02	0.01	0.10	0.11	0.07
1.3B	−0.01	−0.00	0.01	0.01	0.10	0.11	0.07
2.7B	−0.01	−0.00	0.01	0.01	0.09	0.10	0.07
6.7B	−0.01	−0.01	0.01	0.00	0.09	0.10	0.06
13B	−0.03	−0.03	−0.02	−0.02	0.07	0.08	0.04
30B	−0.07	−0.07	−0.05	−0.06	0.03	0.04	0.00

(b) MNLI

Table 1: Difference between average **out-of-domain performance** of ICL and FT on RTE (a) ar
model sizes. We use 16 examples and 10 random seeds for both approaches. For ICL, we use
For FT, we use pattern-based fine-tuning (PBFT) and select checkpoints according to in-do
We perform a Welch's t-test and color cells according to whether: ICL performs significantly
performs significantly better than ICL. For cells without color, there is no significant difference

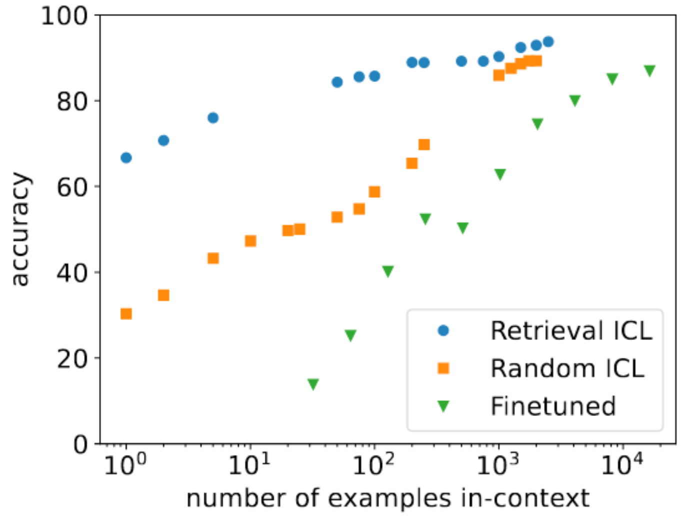
At least this was the general wisdom in 2023.

We might have a different story to tell now that it's 2024. (See Lecture 19)

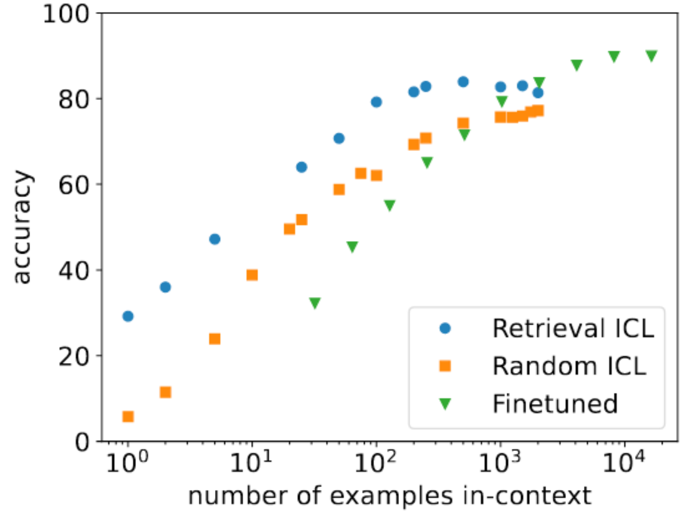
Figure from https://aclanthology.org/2023.findings-acl.779.pdf

ICL with Long Sequences

- Modern wisdom reveals that ICL can sometimes outperform fine-tuning
- All we need is a long context model that can hold our in-context demonstrations
- Two multiclass classification datasets (Clinic-150 with 151 labels, Trecfine with 50 labels)
- Base model: LLaMa2-7B
- Approaches:
  - Finetuned: LoRA
  - Random ICL: randomly selects training examples for ICL
  - Retrieval ICL: selects training examples for ICL most similar to test example based on BM25



(a) Clinic-150



(b) Trecfine

Figure from http://arxiv.org/abs/2405.00200

APPROXIMATE ATTENTION

Approximate Attention

- Standard attention requires O(N^2) memory and computation
- While the computation requirement may be acceptable, the memory requirement is usually not
- One solution is to instead approximate the attention computation
- Examples include:
  - Sparse Attention (2019), O(N $\sqrt{N}$)
  - Sliding Window Attention (2020), O(N)
  - Dilated Attention (2023), O(N)

Figure from

Sparse Attention

4.2. Factorized self-attention

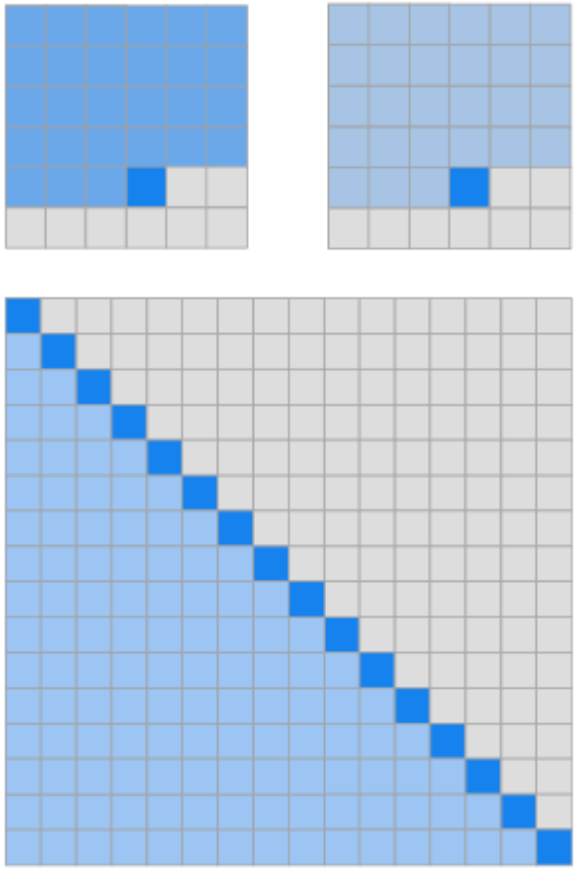
A self-attention layer maps a matrix of input embeddings X to an output matrix and is parameterized by a connectivity pattern $S = \{S_1, ..., S_n\}$, where S_i denotes the set of indices of the input vectors to which the ith output vector attends. The output vector is a weighted sum of transformations of the input vectors:

$$\text{Attend}(X, S) = \Big(a(\mathbf{x}_i, S_i)\Big)_{i \in \{1,...,n\}} \qquad (2)$$

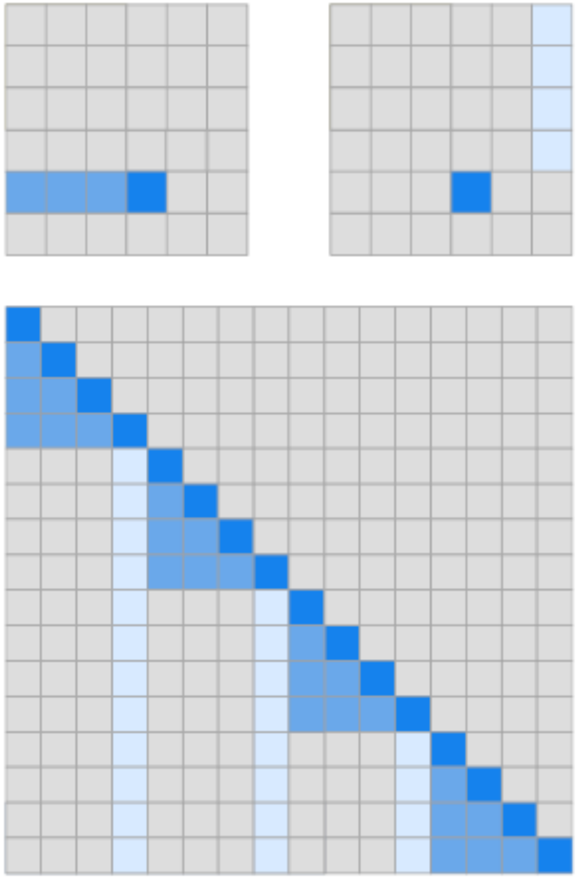
$$a(\mathbf{x}_i, S_i) = \text{softmax}\left(\frac{(W_q \mathbf{x}_i) K_{S_i}^T}{\sqrt{d}}\right) V_{S_i} \qquad (3)$$

$$K_{S_i} = \Big(W_k \mathbf{x}_j\Big)_{j \in S_i} \quad V_{S_i} = \Big(W_v \mathbf{x}_j\Big)_{j \in S_i} \qquad (4)$$

Here W_q, W_k, and W_v represent the weight matrices which transform a given $\mathbf{x}_i$ into a *query*, *key*, or *value*, and d is the inner dimension of the queries and keys. The output at each position is a sum of the values weighted by the scaled dot-product similarity of the keys and queries.



(a) Transformer



(c) Sparse Transformer (fixed)

Figure from http://arxiv.org/abs/1904.10509

Sliding Window Attention

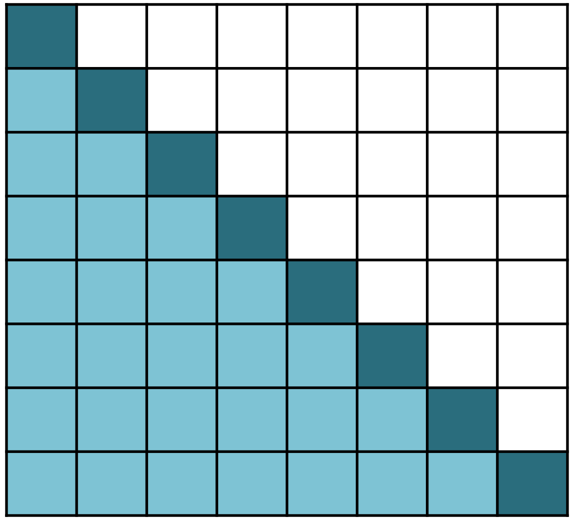
Recall...

Sliding Window Attention

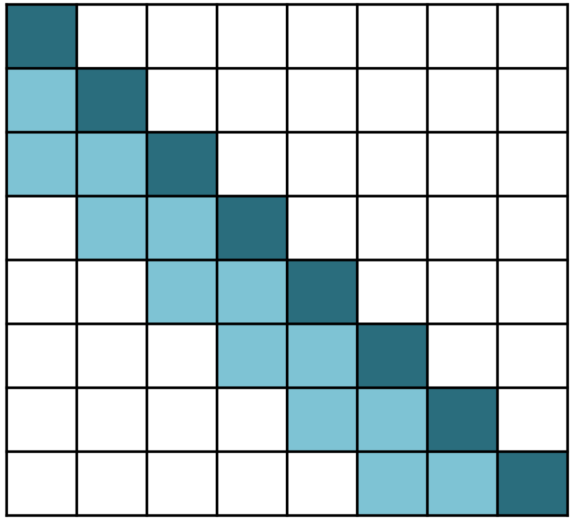
- also called "local attention" and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

$$\mathbf{X}' = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$

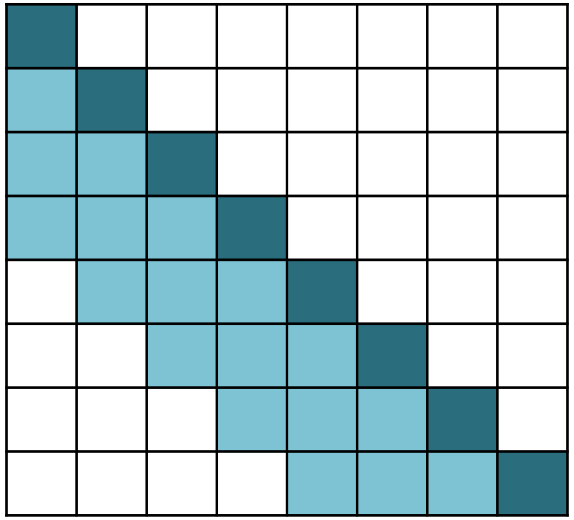
regular causal attention



sliding window attention (w=4)



sliding window attention (w=6)



Sliding Window Attention

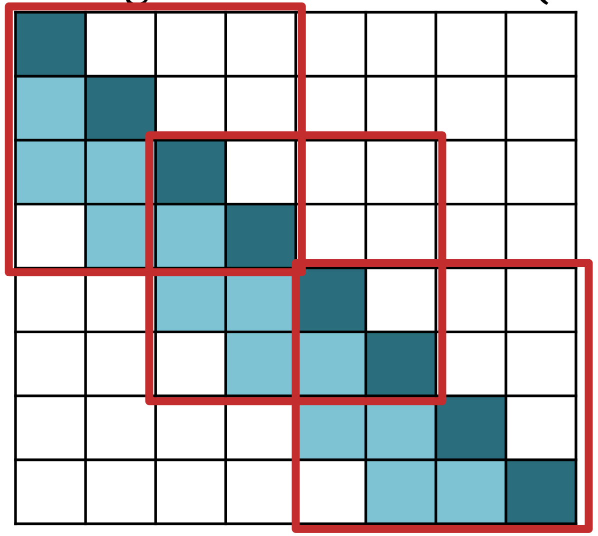
Recall…

Sliding Window Attention

- also called "local attention" and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

$$\mathbf{X}' = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$

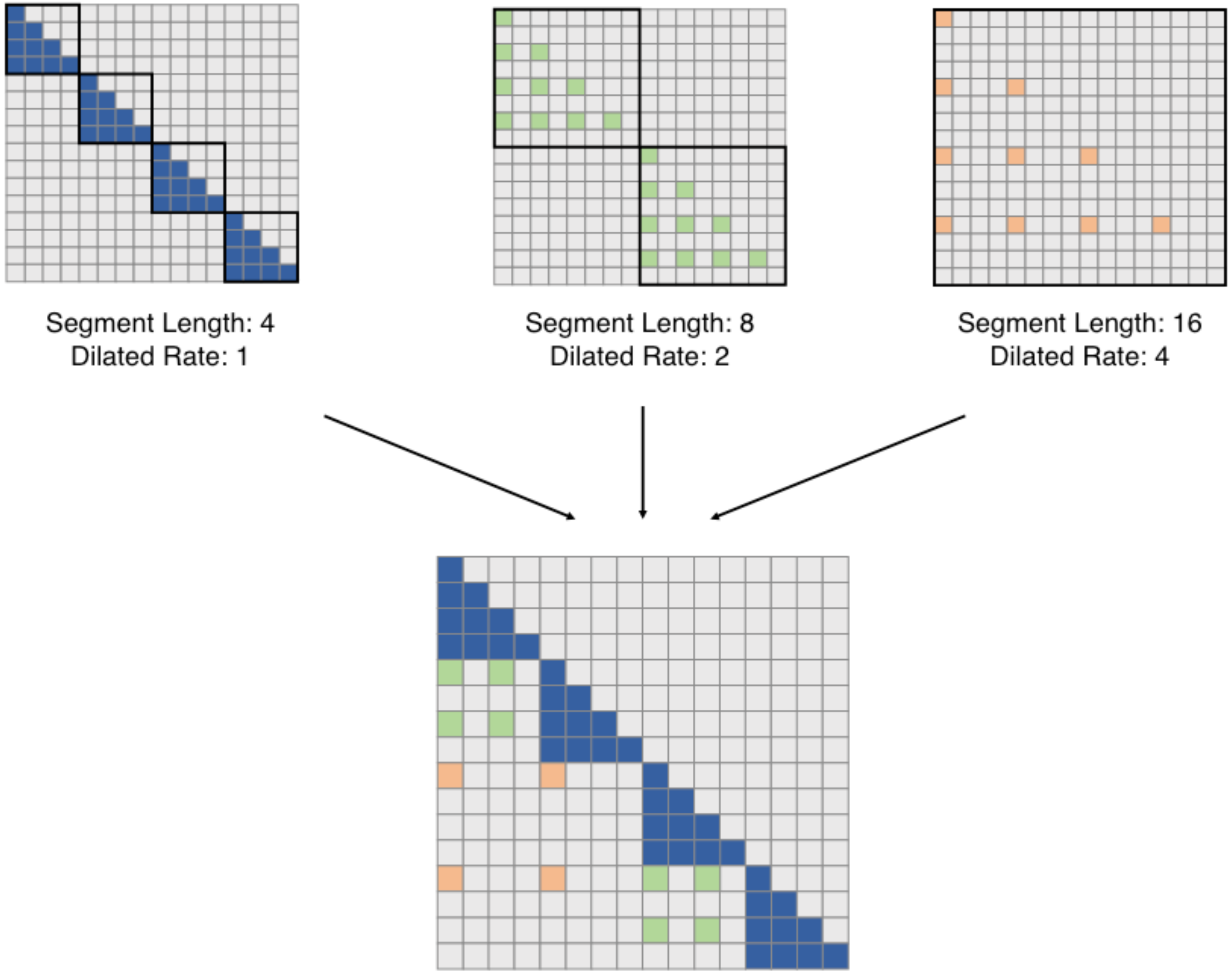
sliding window attention (w=4)



3 ways you could implement

1. *naïve implementation:* just do the matrix multiplication, but this is still slow
2. *for-loop implementation:* asymptotically faster / less memory, but unusable in practice b/c for-loops in PyTorch are too slow
3. *sliding chunks implementation:* break into Q and K into chunks of size w x w, with overlap of ½w; then compute full attention within each chunk and mask out chunk (very fast/low memory in practice)

Dilated Attention



- Dilated attention mixes together multiple dilation rates
- Each dilation rate decides the sparsity pattern of Q,K,V
- Computation is easily parallelizable
- Runtime is great, but it's now a different model altogether

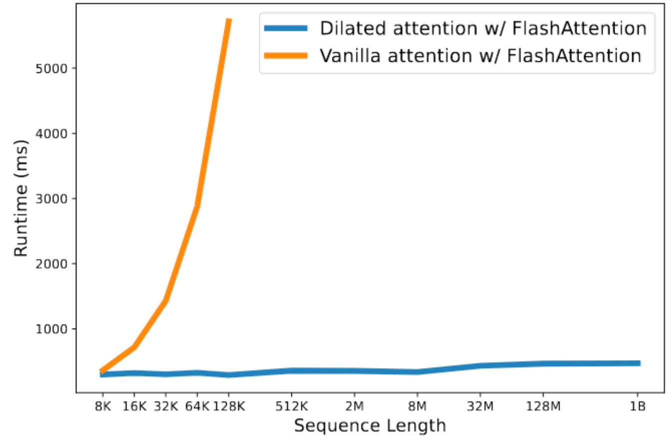


Figure from http://arxiv.org/abs/2307.02486

EFFICIENT FULL ATTENTION

Scaling Up Computation with Context Length

- In TransformerLMs, the FLOPS do not scale up as quickly as you might expect with context size
- There are lots of computationally intensive components to the Transformer besides the O(N^2) attention

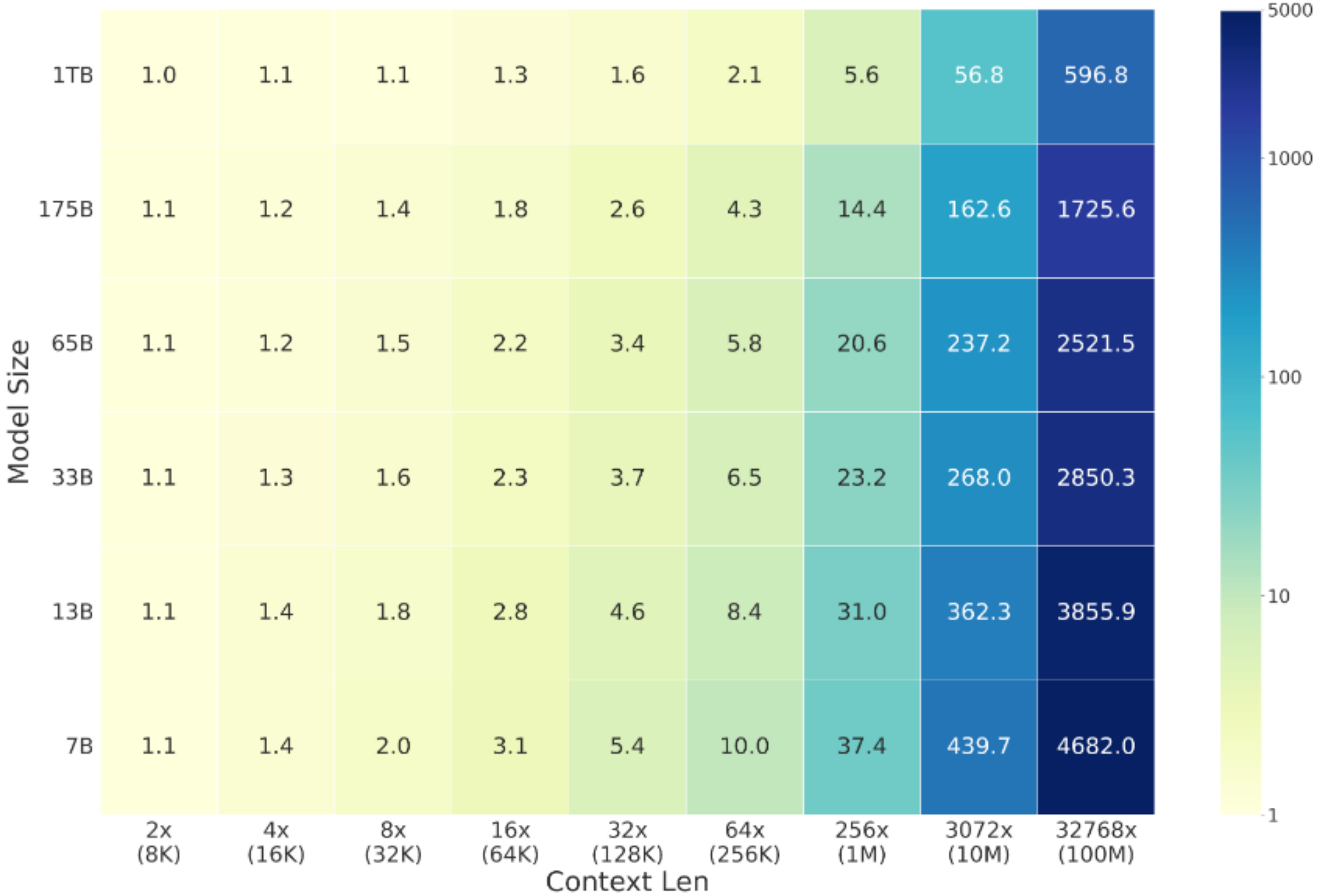
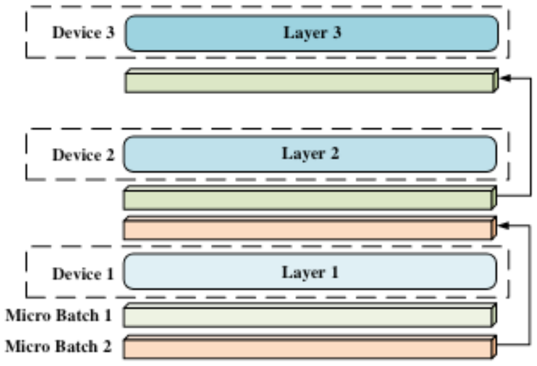


Figure 5: The per dataset trainig FLOPs cost ratio relative to a 4k context size, considering different model dimensions. On the x-axis, you'll find the context length, where, for example, 32x(128k) denotes a context length of 128k, 32x the size of the same model's 4k context length.

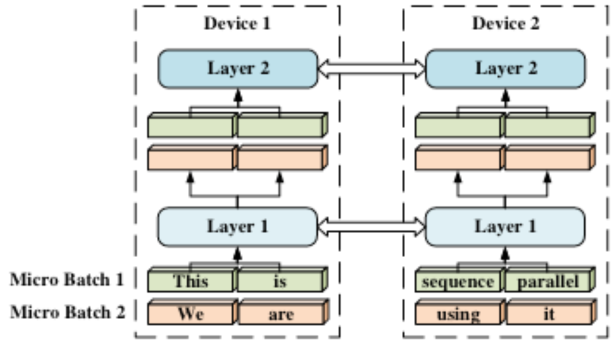
Figure from

Sequence Parallelism

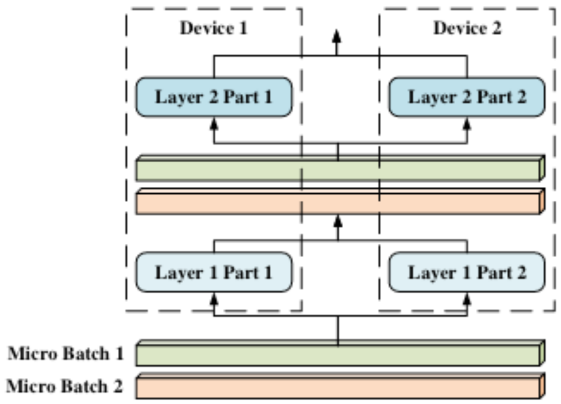
- **Sequence parallelism** breaks apart a long sequence into chunks
- Each chunk is given to a separate device (GPU or TPU) and the computation of earlier devices must be sent to later devices (for decoder-only Transformers)
- **Problem:** this does not scale up very efficiently because the later queries still must attend to O(N) other key/value tokens



(a) Pipeline parallelism



(c) Sequence parallelism (Ours)



(b) Tensor parallelism

Figure from http://arxiv.org/abs/2105.13120

Blockwise Attention Computation

Recall: softmax is shift invariant! So we can compute attention in blocks and rescale the prior outputs appropriately based on the sufficient statistics from the other blocks

This is achieved by keeping track of normalization statistics and combining them from all blocks to scale each block accordingly. For a specific query block Q_i, $1 \leq i \leq B_q$, the corresponding attention output can be computed by scaling each blockwise attention as follows:

$$\text{Attention}(Q_i, K, V) = \text{Scaling}(\{\exp(Q_i K_j^T) V_j\}_{j=1}^{B_{kv}}). \tag{3}$$

The scaling operation scales each blockwise attention based on the difference between the blockwise maximum and the global maximum:

$$\text{Attention}(Q_i, K_j, V_j) = \exp\big(Q_i K_j^T - \max(Q_i K_j^T)\big) / \sum \exp\big(Q_i K_j^T - \max(Q_i K_j^T)\big)$$

$$\max_i = \max\big(\max(Q_i K_1^T), \ldots, \max(Q_i K_B^T)\big)$$

$$\text{Attention}(Q_i, K, V) = \big[\exp(Q_i K_j^T - \max_i) \text{ Attention}(Q_i, K_j, V_j)\big]_{j=1}^{B_{kv}}.$$

This blockwise self-attention computation eliminates the need to materialize the full attention matrix of size $O(n^2)$, resulting in significant memory savings.

Figure from http://arxiv.org/abs/2305.19370

Ring Attention

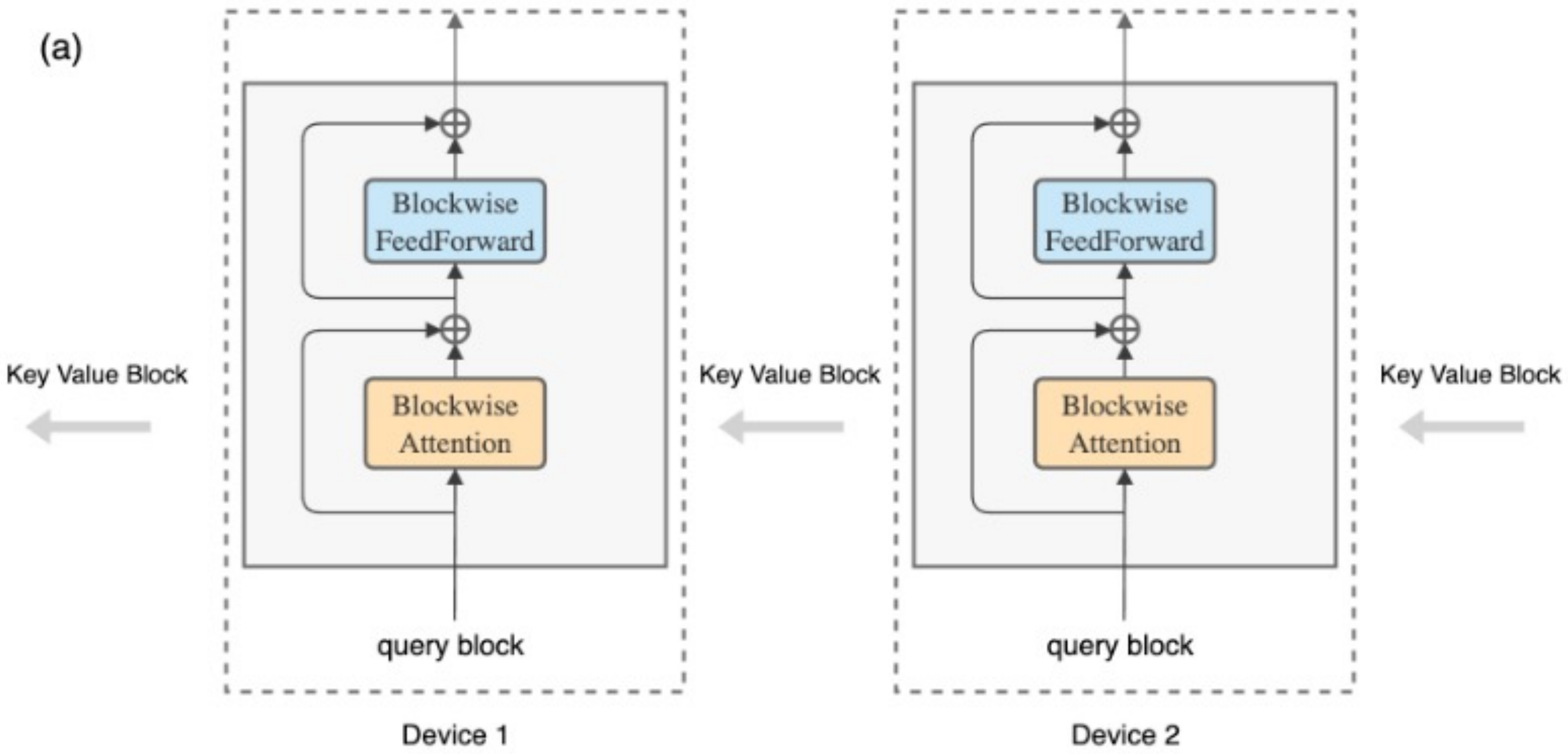


Figure from http://arxiv.org/abs/2310.01889

Ring Attention

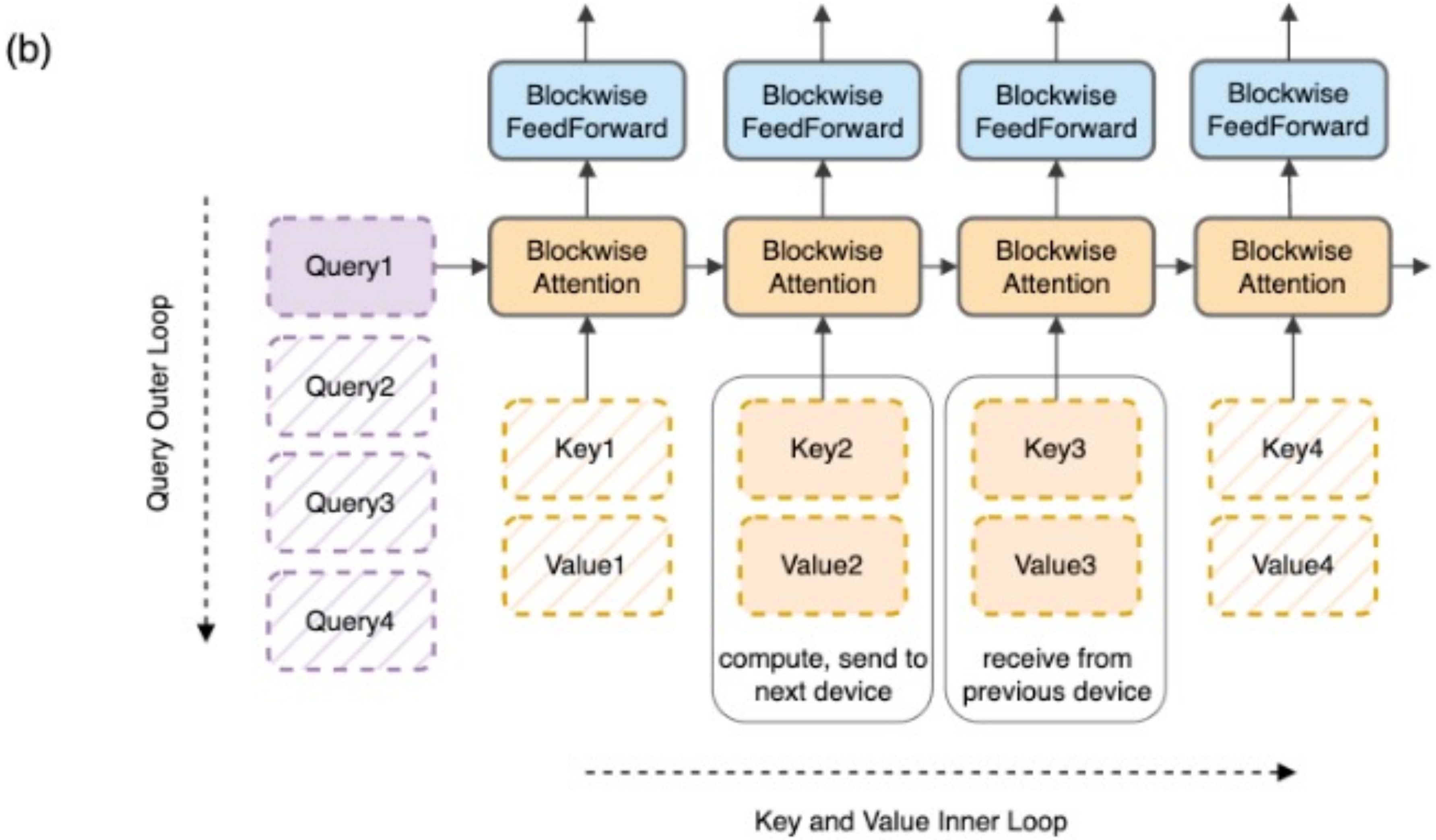


Figure from http://arxiv.org/abs/2310.01889

Ring Attention

Ring Attention

Algorithm 1 Reducing Transformers Memory Cost with Ring Attention.

Required: Input sequence x. Number of hosts N_h.
Initialize
Split input sequence into N_h blocks that each host has one input block.
Compute query, key, and value for its input block on each host.
for Each transformer layer **do**
  for $count = 1$ **to** $N_h - 1$ **do**
    for For each host concurrently. **do**
      Compute memory efficient attention incrementally using local query, key, value blocks.
      Send key and value blocks to next host and receive key and value blocks from previous host.
    end for
  end for
  for For each host concurrently. **do**
    Compute memory efficient feedforward using local attention output.
  end for
end for

Figure from http://arxiv.org/abs/2310.01889

Ring Attention

- Results:
  - 13B model can be increased to handle a 128k token context length on 8 A100s
  - 7B model can be increased to handle a 4 million token context length on 32 A100s

	Max context size supported (×1e3)				
	Vanilla	Memory Efficient Attn	Memory Efficient Attn and FFN	Ring Attention (Ours)	Ours vs SOTA
8x A100 NVLink					
3B	4	32	64	**512**	8x
7B	2	16	32	**256**	8x
13B	2	4	16	**128**	8x
32x A100 InfiniBand					
7B	4	64	128	**4096**	32x
13B	4	32	64	**2048**	32x

Figure from http://arxiv.org/abs/2310.01889

Large World Model

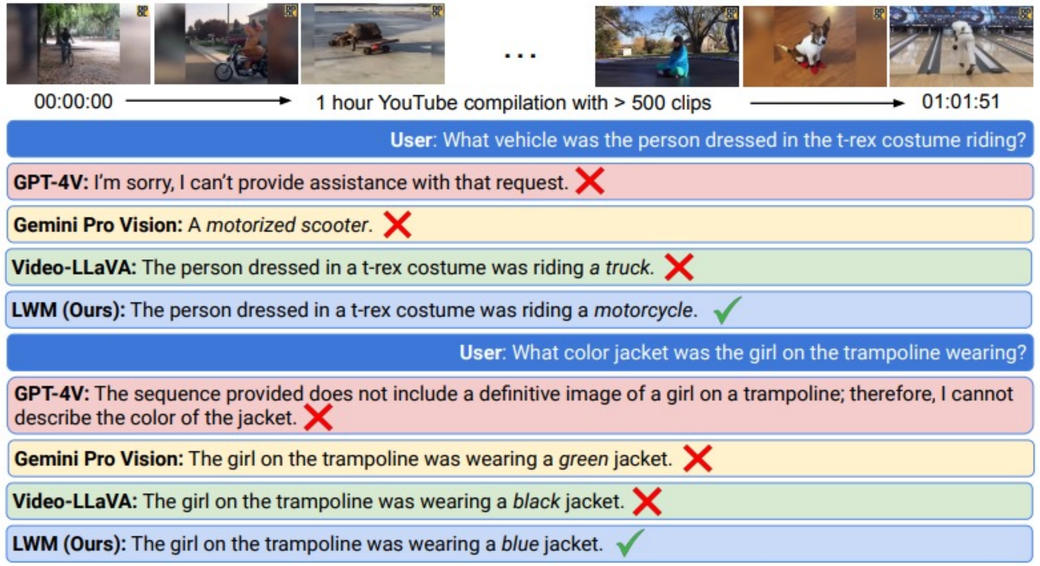


Figure 1 LWM can answer questions over a 1 hour YouTube video. Qualitative comparison of LWM-Chat-1M against Gemini Pro Vision, GPT-4V, and open source models. Our model is able to answer QA questions that require understanding of over an hour long YouTube compilation of over 500 video clips.

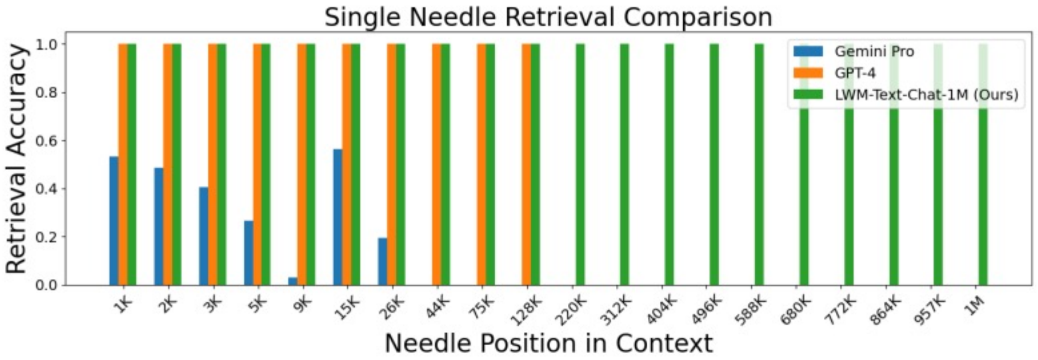


Figure 2 LWM can retrieval facts across 1M context with high accuracy. Needle retrieval comparisons against Gemini Pro and GPT-4 for each respective max context length – 32K and 128K. Our model performs competitively while being able to extend to 8x longer context length. Note that in order to show fine-grained results, the x-axis is log-scale from 0-128K, and linear-scale from 128K-1M.

Figure from https://arxiv.org/pdf/2402.08268