



Dynamic Sparse Attention at Scale

Marcos V. Treviso

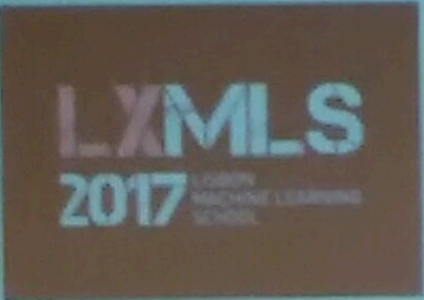
Assist. Professor at IST / Univ. of Lisbon

July 24

9 years ago at LxMLS 2017



Cho was here presenting the "attention mechanism" (before transformers!!!)



LXMLS
2017 LONDON
MACHINE LEARNING
SCHOOL

Neural Machine Translation and Beyond

Kyunghyun Cho
Courant Institute & Center for Data Science, New York University
Facebook AI Research

Reasoning in LLMs

Standing in for Ivan Titov — “Post-training for Reasoning in LLMs”

Different talk, but **same bottleneck**



Reasoning in LLMs

Standing in for Ivan Titov — “Post-training for Reasoning in LLMs”

Different talk, but **same bottleneck**



Long chains of thought

LLMs can think for
minutes before answering



Long tool traces

agents read repos, logs, docs...
everything is context

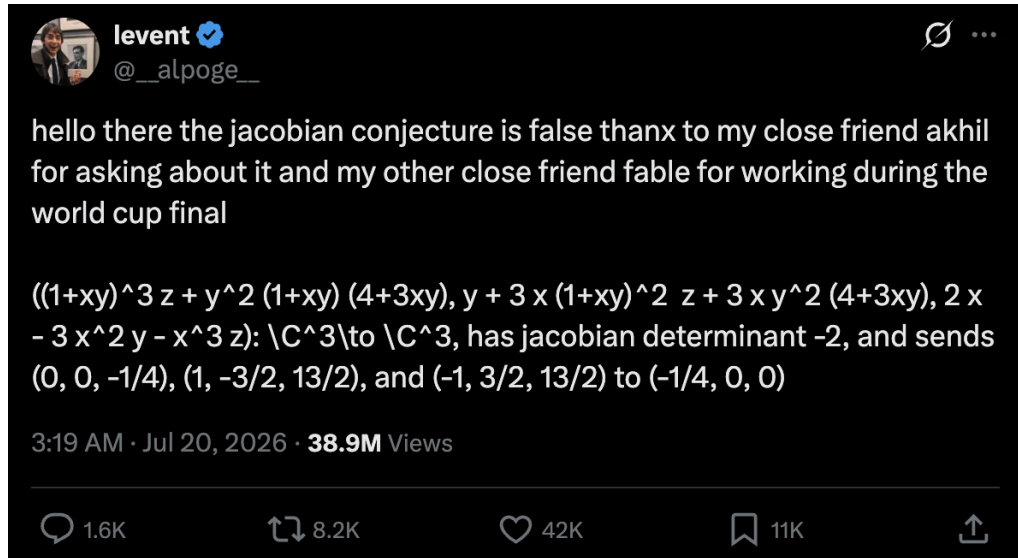


Long sessions

multi-hour conversations
that never reset

Reasoning in LLMs

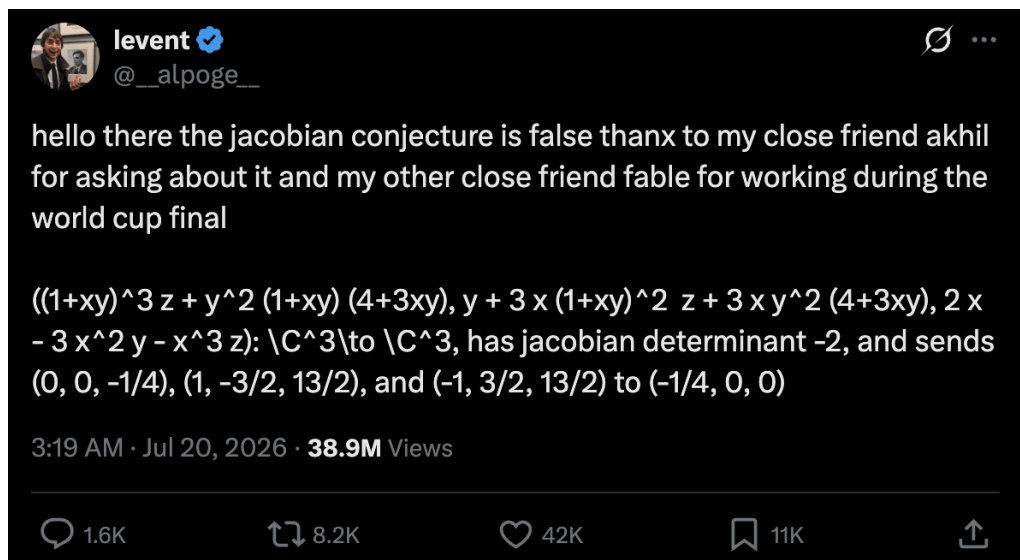
Claude Fable 5 discovered a counterexample to an 87-year-old math problem (Jacobian conjecture)



https://x.com/__alpoge__/status/2079028340955197566

Reasoning in LLMs

Claude Fable 5 discovered a counterexample to an 87-year-old math problem (Jacobian conjecture)



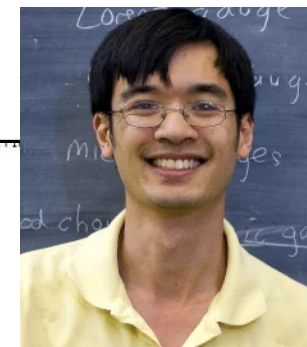
https://x.com/__alpoge__/status/2079028340955197566

A digestion of the Jacobian conjecture counterexample

Posted by [Terence Tao](#)

The notorious [Jacobian conjecture](#) can be formulated concretely over the complex numbers as follows.

Conjecture 1 (Jacobian Conjecture) Let $F : \mathbb{C}^n \rightarrow \mathbb{C}^n$ be a polynomial map in n complex variables, whose Jacobian $\det DF$ is a non-zero constant. Then F is invertible (with



<https://terrytao.wordpress.com/2026/07/21/a-digestion-of-the-jacobian-conjecture-counterexample/>

AI disclosure: I [used an AI chatbot](#) to discuss various aspects of this problem and to confirm several of the calculations made here.

Terence Tao vs GPT 5.6

- 10–20 min of thinking per hard question
- a chat so long my browser lags scrolling it
- after some point, he "activated Pro" for it

Category	Estimate
Cumulative input	≈ 2.9M
Reasoning tokens	≈ 95K
Visible output	≈ 110K
Total usage	≈ 3.1M

* estimated with tiktoken

Terence Tao vs GPT 5.6

- 10–20 min of thinking per hard question
- a chat so long my browser lags scrolling it
- after some point, he "activated Pro" for it

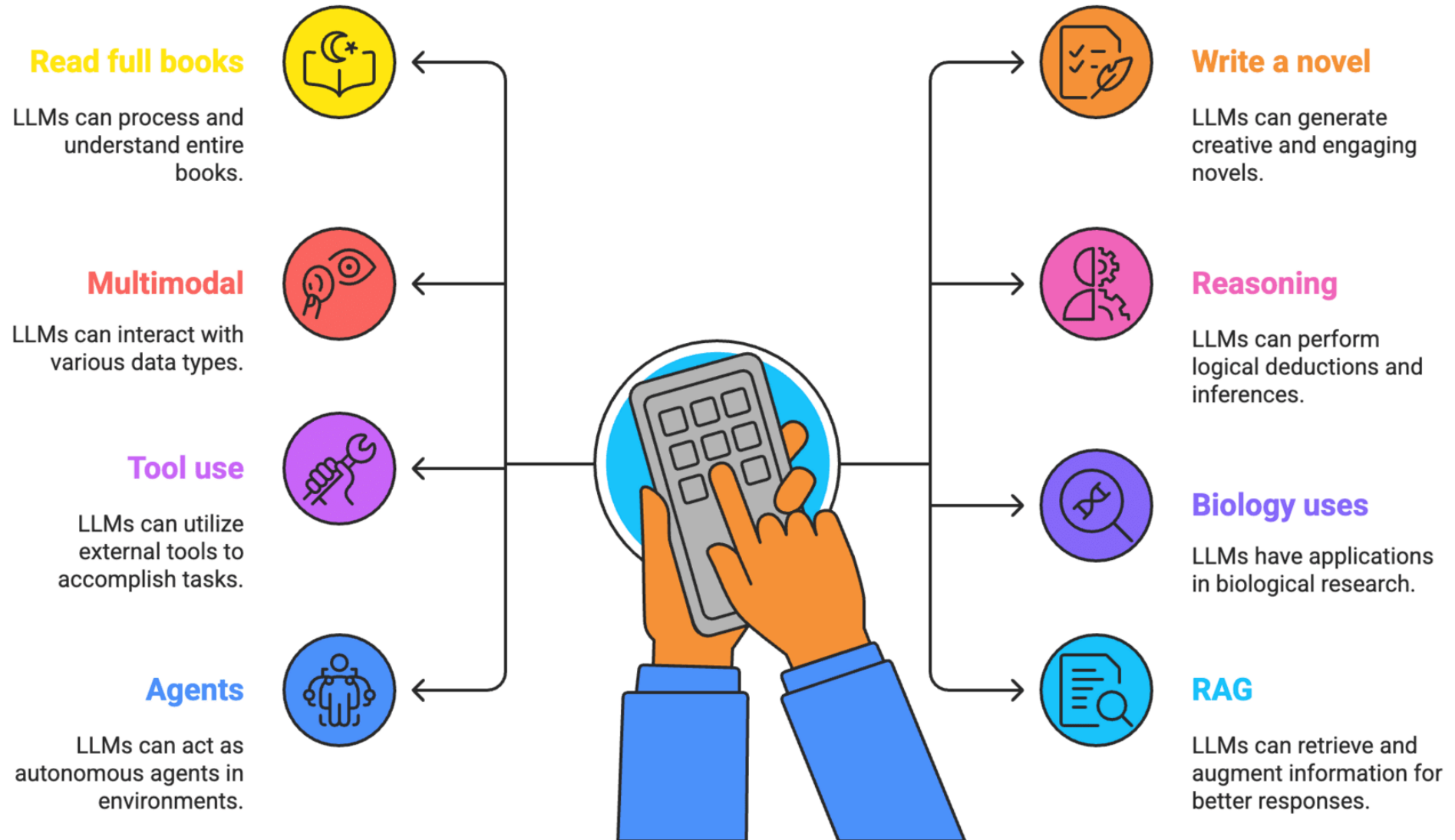
Category	Estimate
Cumulative input	≈ 2.9M
Reasoning tokens	≈ 95K
Visible output	≈ 110K
Total usage	≈ 3.1M

* estimated with tiktoken

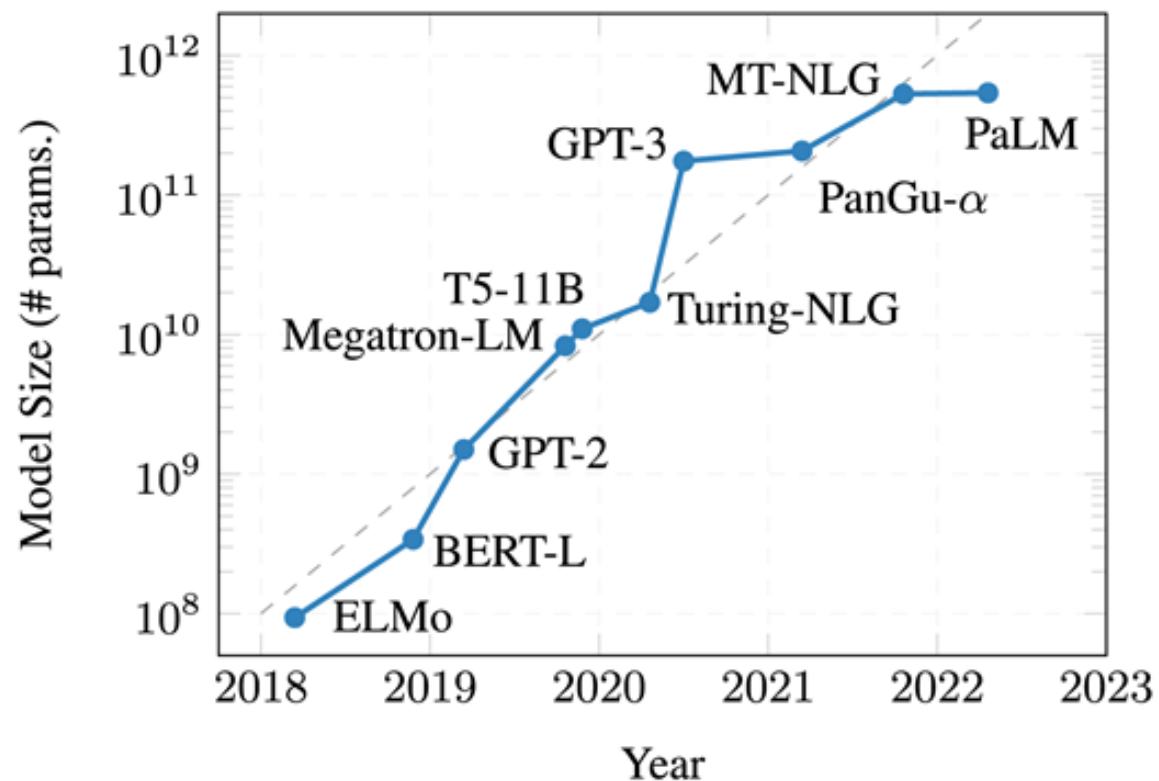
Frontier reasoning = extremely long inputs + extremely long generations

Reasoning is bottlenecked by **long context**
...and, in LLMs, long context is bottlenecked by **attention**

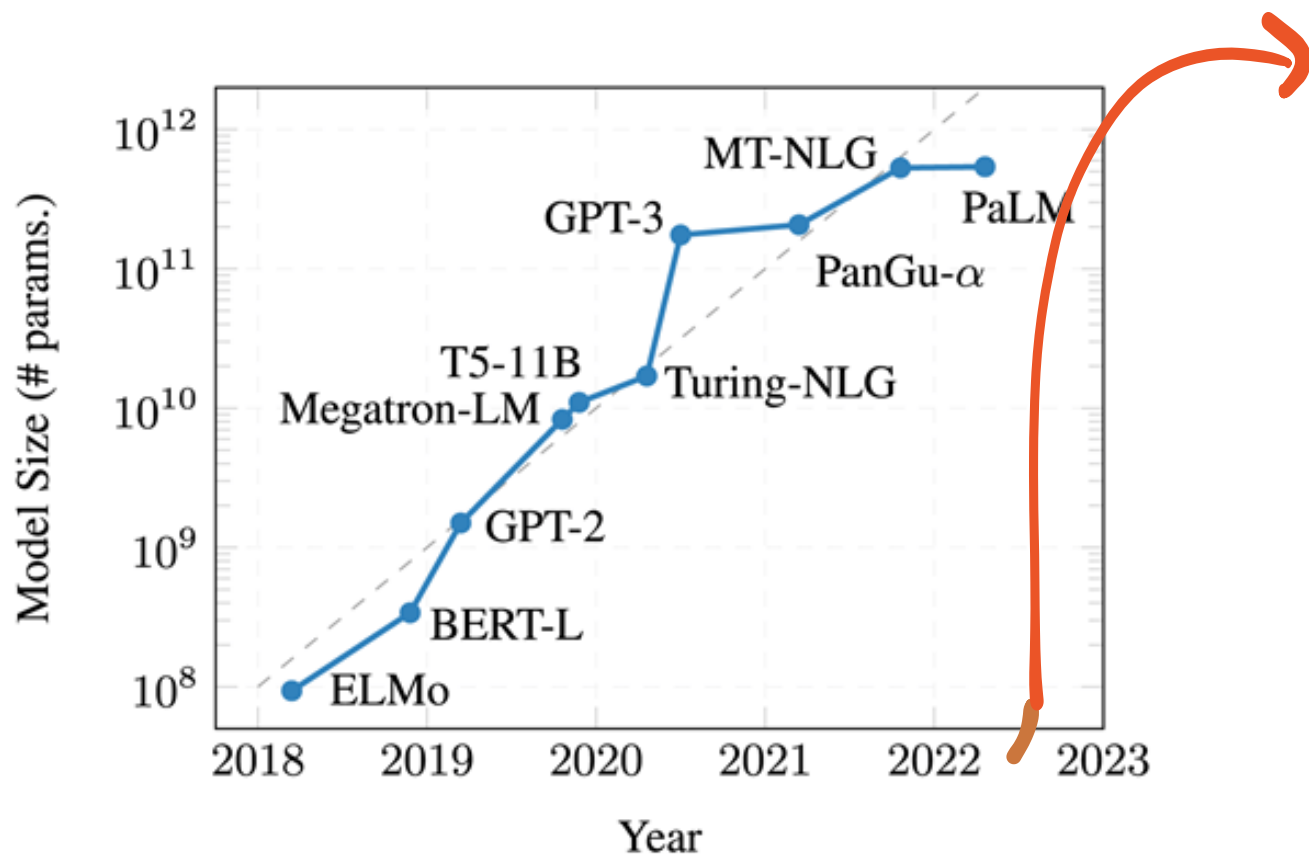
Capabilities



From "Transformers" to LLMs

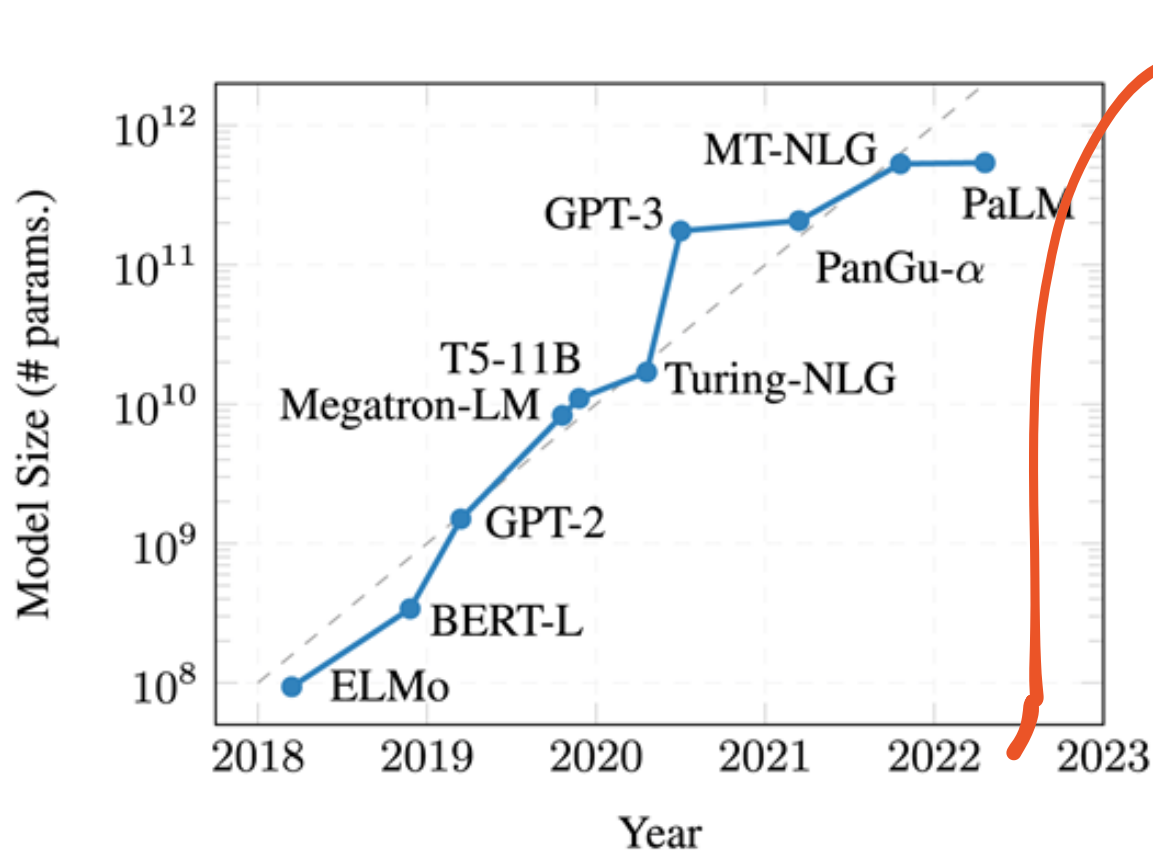


From "Transformers" to LLMs



ChatGPT was released in Nov. 30, 2022

From "Transformers" to LLMs

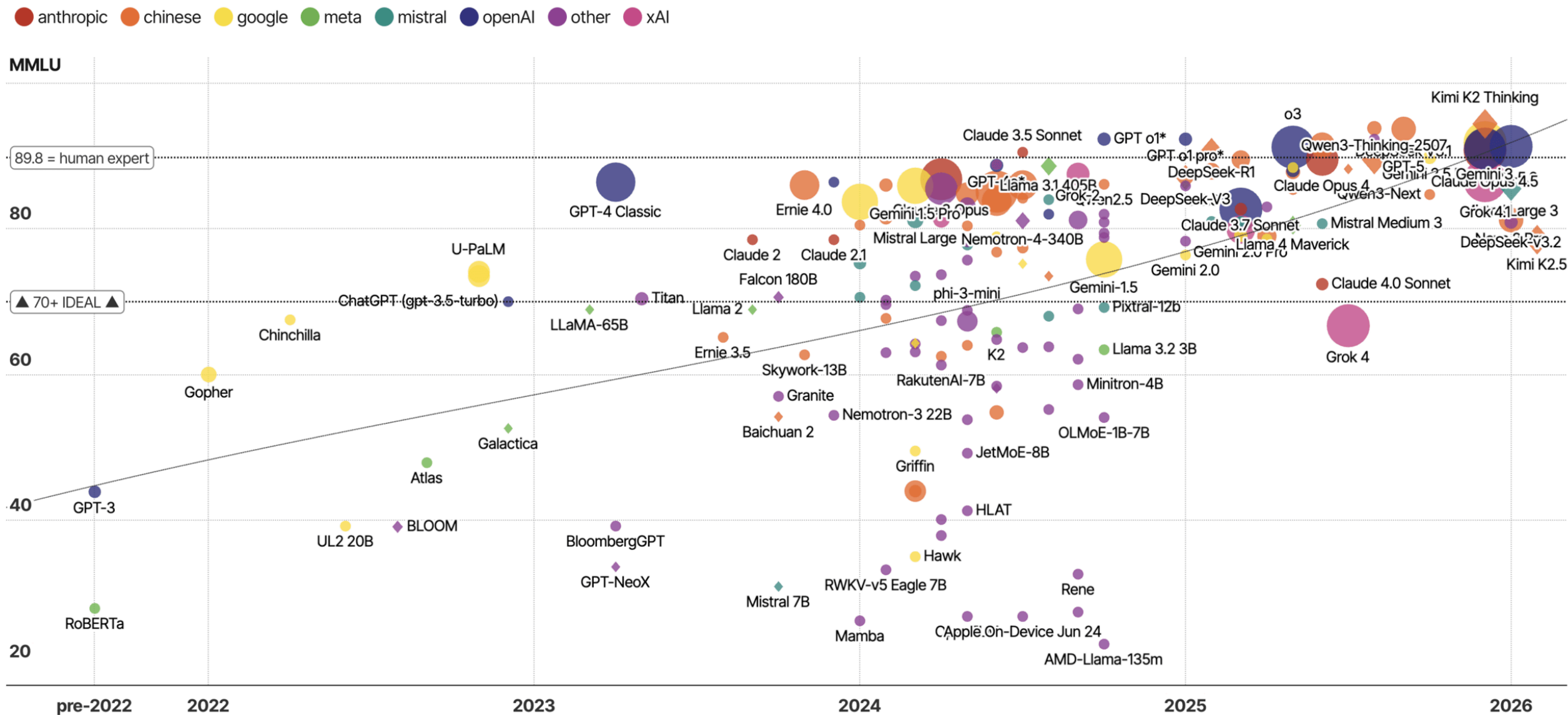


ChatGPT was released in Nov. 30, 2022

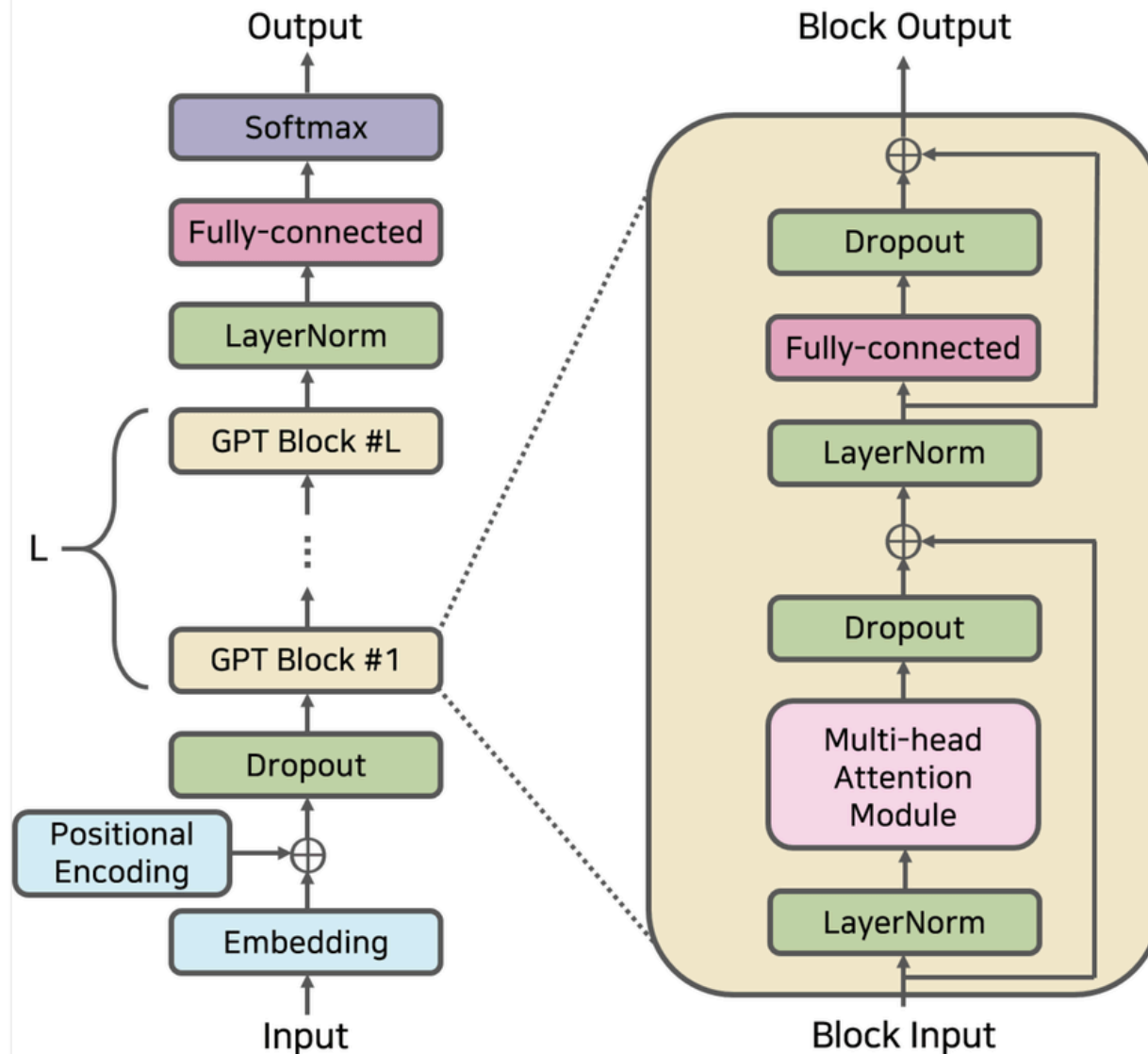
Large Language Models (LLMs) era:

- Trained on "chat" data
- LLaMa, Gemini, Claude, ...
- Next token prediction
- Billions to Trillions of parameters

LLM performance and size evolution



Generative Pretrained *Transformer* (GPT)



Lecture 4:
Transformers and LLMs
[-> youtube](#)



Self-Attention

queries

$$\mathbf{Q} \in \mathbb{R}^{n \times d}$$

keys

$$\mathbf{K} \in \mathbb{R}^{n \times d}$$

values

$$\mathbf{V} \in \mathbb{R}^{n \times d}$$

1. Compute pairwise scores

$$\mathbf{Z} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d} \in \mathbb{R}^{n \times n}$$

2. Map rows to probabilities

$$\mathbf{P} = \text{softmax}(\mathbf{Z}) \in \mathbb{R}^{n \times n}$$

3. Convex combination

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

Self-Attention

queries

$$\mathbf{Q} \in \mathbb{R}^{n \times d}$$

keys

$$\mathbf{K} \in \mathbb{R}^{n \times d}$$

values

$$\mathbf{V} \in \mathbb{R}^{n \times d}$$

1. Compute pairwise scores

$$\mathbf{Z} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d} \in \mathbb{R}^{n \times n}$$

2. Map rows to probabilities

$$\mathbf{P} = \text{softmax}(\mathbf{Z}) \in \mathbb{R}^{n \times n}$$

3. Convex combination

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

```
1 Z = torch.matmul(Q, K.transpose(-1, -2))  
2 P = torch.softmax(Z, dim=-1)  
3 O = torch.matmul(P, V)
```

Self-Attention

queries

$$\mathbf{Q} \in \mathbb{R}^{n \times d}$$

keys

$$\mathbf{K} \in \mathbb{R}^{n \times d}$$

values

$$\mathbf{V} \in \mathbb{R}^{n \times d}$$

1. Compute pairwise scores

$$\mathbf{Z} = \mathbf{Q}\mathbf{K}^\top / \sqrt{d} \in \mathbb{R}^{n \times n}$$

2. Map rows to probabilities

$$\mathbf{P} = \text{softmax}(\mathbf{Z}) \in \mathbb{R}^{n \times n}$$

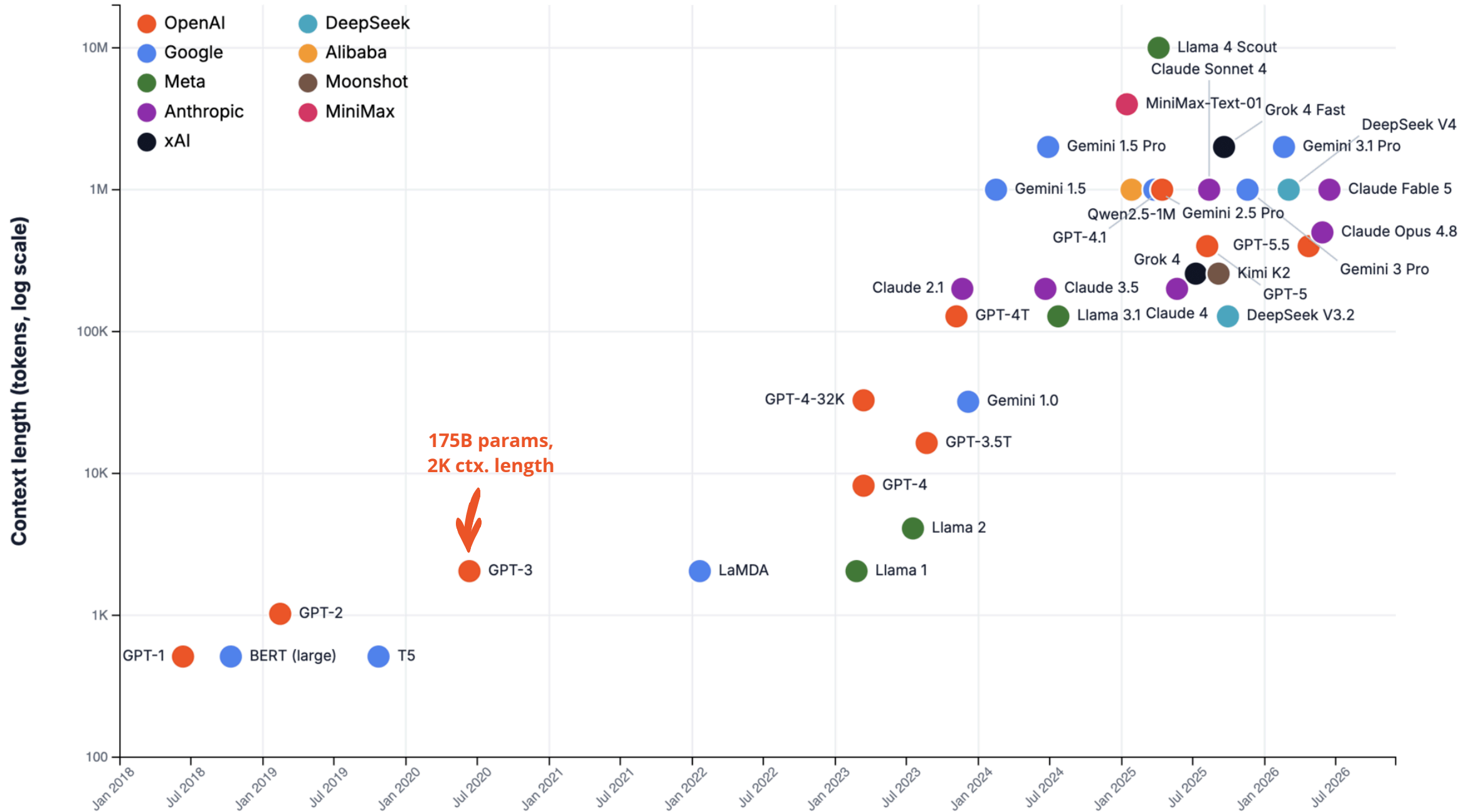
3. Convex combination

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

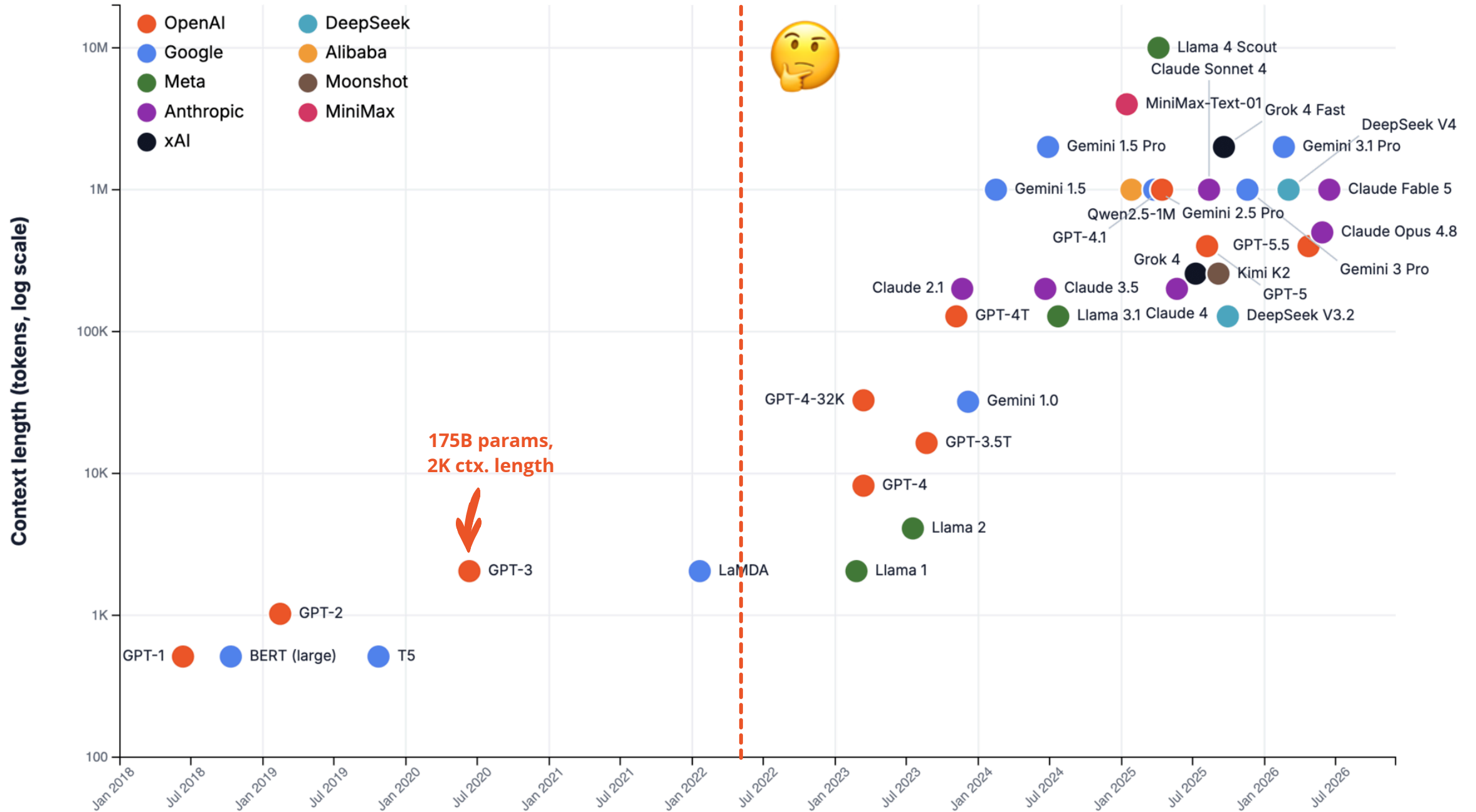
```
1 Z = torch.matmul(Q, K.transpose(-1, -2))  
2 P = torch.softmax(Z, dim=-1)  
3 O = torch.matmul(P, V)
```

Where is the bottleneck?

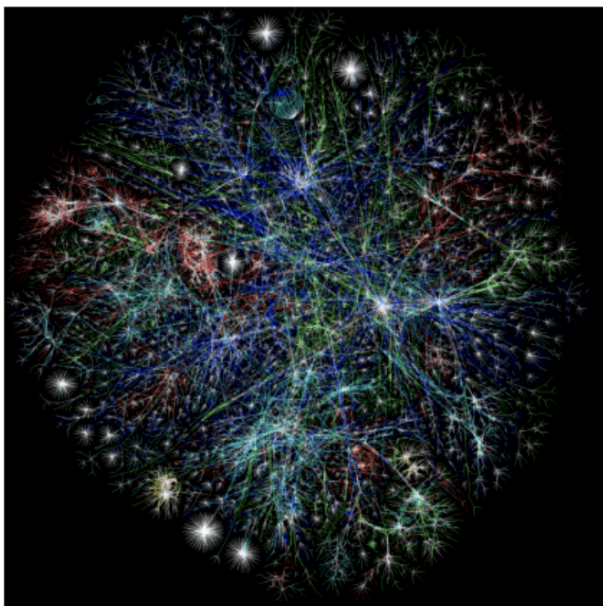
LLM Context Length Evolution (2018–2026)



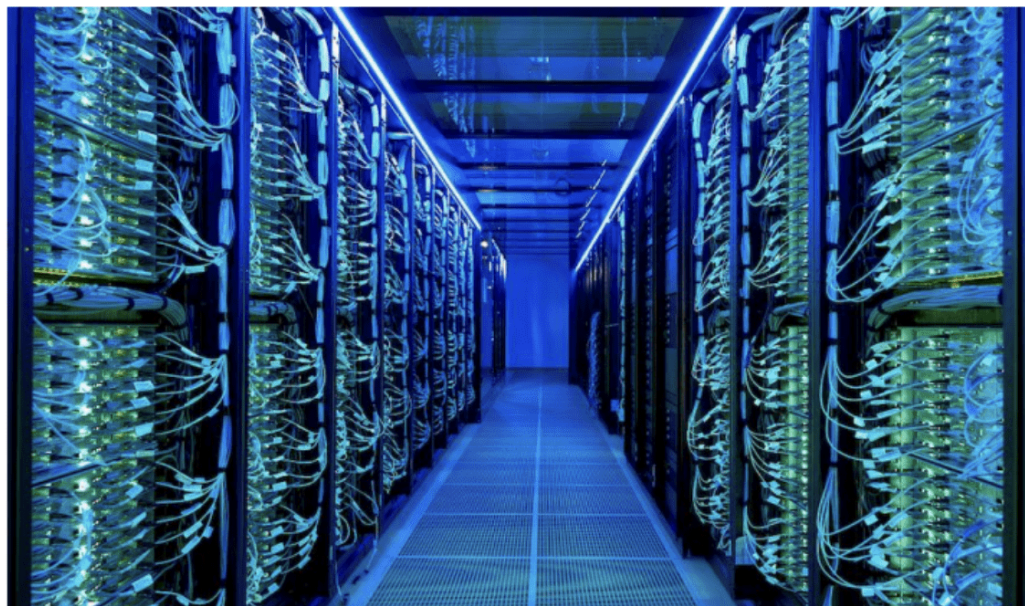
LLM Context Length Evolution (2018–2026)



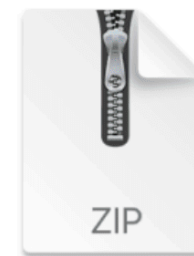
Scaling is challenging



Chunk of the internet,
~10TB of text



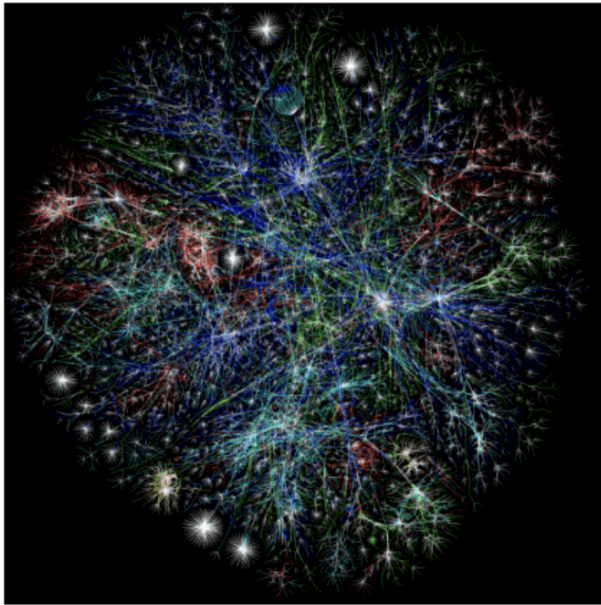
6,000 GPUs for 12 days,
~1e24 FLOPS



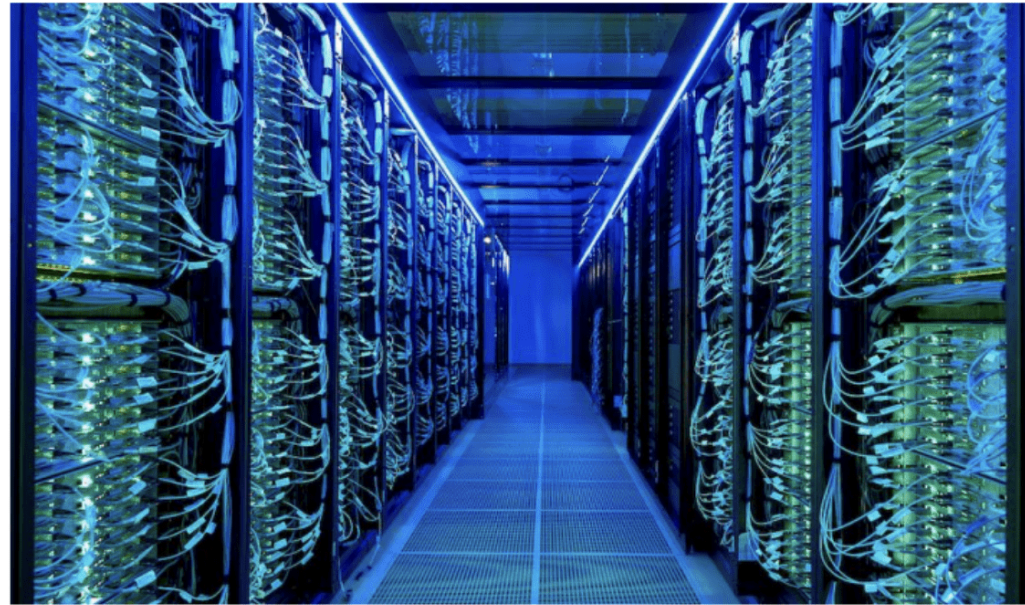
parameters.zip

~140GB file

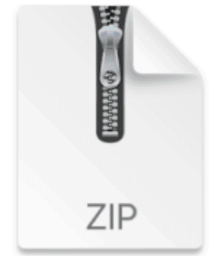
Scaling is challenging



Chunk of the internet,
~10TB of text



6,000 GPUs for 12 days,
~1e24 FLOPS



parameters.zip

~140GB file

The estimated cost? ~2M\$ 💰💰💰
for a single training run

* Numbers for LLaMA-2 70B

Slide by Andrej Karpathy

Scaling is challenging

Training Complexity

- Time: $\mathcal{O}(n^2 \times d)$
- Memory: $\mathcal{O}(n^2)$

Inference (with KV cache)

- Time: $\mathcal{O}(n \times d)$
- Memory: $\mathcal{O}(n \times d)$

Scaling is challenging

Training Complexity

- Time: $\mathcal{O}(n^2 \times d)$
- Memory: $\mathcal{O}(n^2)$

Inference (with KV cache)

- Time: $\mathcal{O}(n \times d)$
- Memory: $\mathcal{O}(n \times d)$

Scaling is challenging

Training Complexity

- Time: $\mathcal{O}(n^2 \times d)$
- Memory: $\mathcal{O}(n^2)$

Inference (with KV cache)

- Time: $\mathcal{O}(n \times d)$
- Memory: $\mathcal{O}(n \times d)$

$B = 1$ sample

$n = 4096$ tokens

$H = 16$ heads

32 float precision



$4096^2 \times 16 \times 32/8 = \mathbf{1GB}$ + *model params and optimizer states*
for a *single* sample!



Text

 $10^3 - 10^5$

paper $\approx$ 10K · book $\approx$ 150K · codebase $\approx$ millions



Audio

 $10^5 - 10^6$

1 hour $\approx$ 100K–500K tokens, codec-dependent



Video

 $10^6 - 10^7$

1 hour @ 1 fps $\times$ $\sim$ 250 tokens/frame $\approx$ 1M



Agents

 ∞

unbounded... context is the working memory

 **Text**

$10^3 - 10^5$

paper $\approx$ 10K · book $\approx$ 150K · codebase $\approx$ millions

 **Audio**

$10^5 - 10^6$

1 hour $\approx$ 100K–500K tokens, codec-dependent

 **Video**

$10^6 - 10^7$

1 hour @ 1 fps $\times$ $\sim$ 250 tokens/frame $\approx$ 1M

 **Agents**

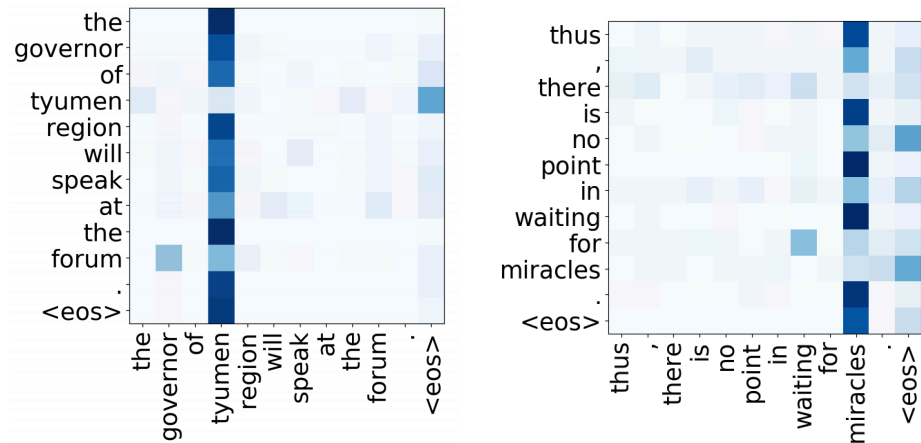
∞

unbounded... context is the working memory

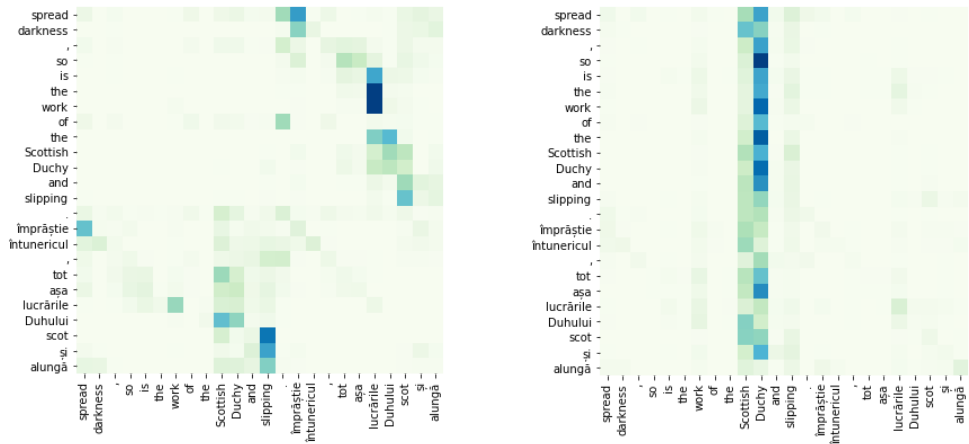
If we want models to handle **long context**,
they will need some kind of **efficient attention**

Before LLMs, attention heads were sparse

(Voita et al., 2019)



(Correia et al., 2019)





Quick poll

🎯 A LLaMA-3 (8B) reads a **128K-token prompt**.

What fraction of the attention-score entries is enough to capture 96% of all attention mass?

A ~50%

B ~25%

C ~3%

D ~0.01%



Quick poll

🎯 A LLaMA-3 (8B) reads a **128K-token prompt**.

What fraction of the attention-score entries is enough to capture 96% of all attention mass?

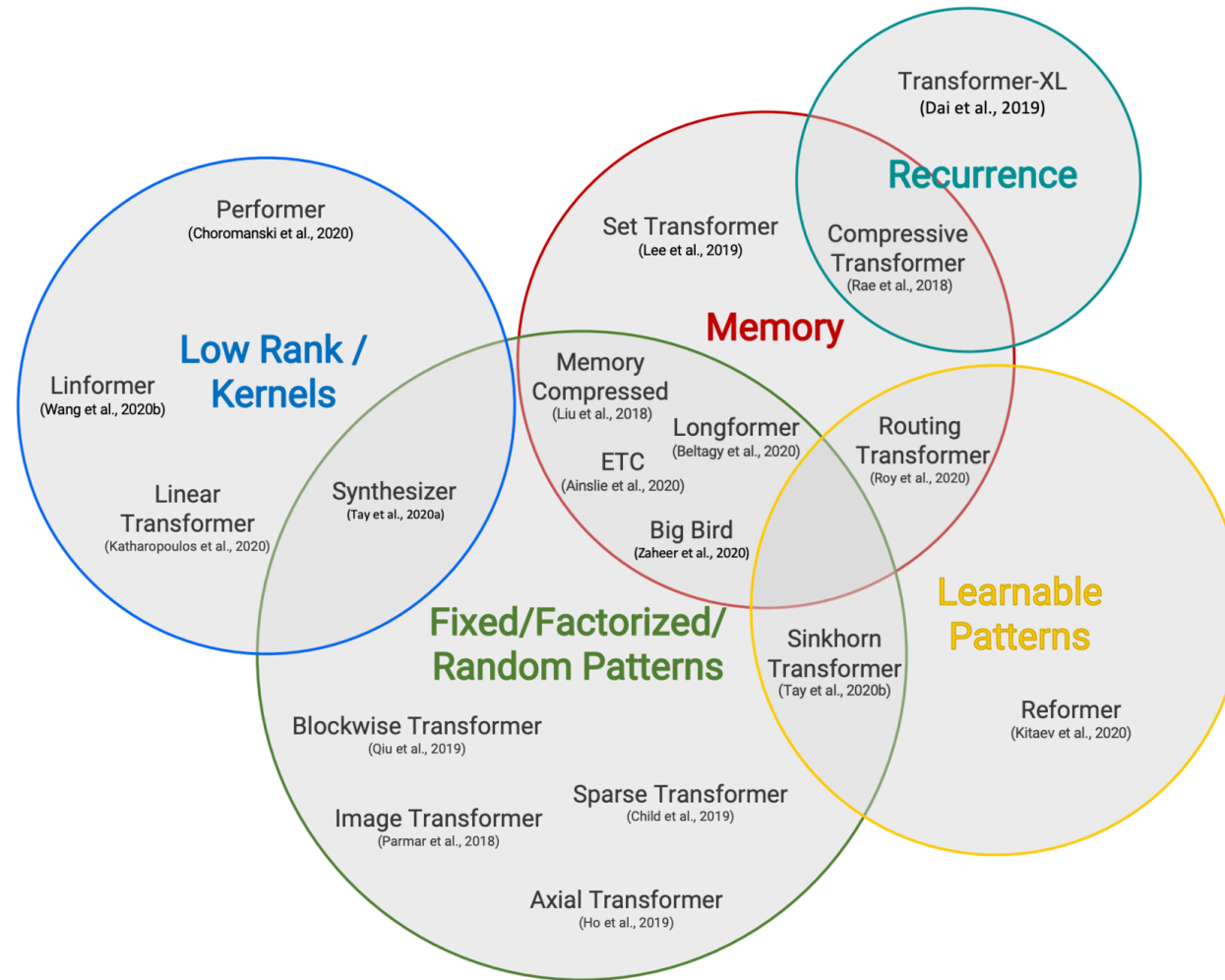
A ~50%

B ~25%

C ~3%

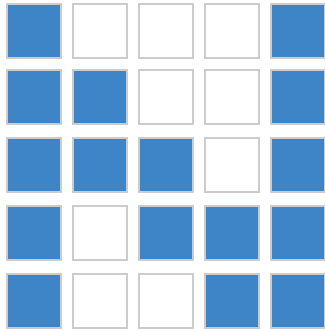
D ~0.01%

Efficient Transformers



Sparse Attention

Fixed Patterns



- **Window:** Local context (Gemma)
- **Global:** Special tokens (Longformer)
- **Random:** Random patterns (BigBird)

😞 Low Adaptivity

😞 Low Flexibility

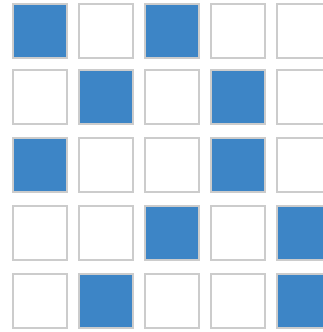
😄 Full Differentiability



$O(n \times k)$ time cost

$O(n \times k)$ memory cost

Dynamic Selection



- **Top-k:** Highest scores (NSA, MoBA, ...)
- **LSH:** Hash similarity (Reformer)
- **Cluster:** Group similarity (Routing)

😄 High Adaptivity

😄 More Flexibility

😞 Approx or No Differentiability

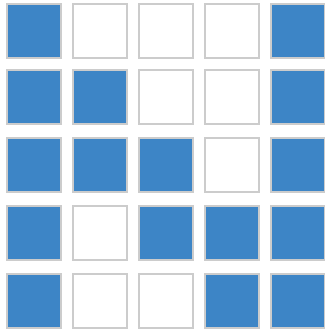


$O(n \log n)$ time cost

$O(n \log n)$ memory cost

Sparse Attention

Fixed Patterns



- **Window:** Local context (Gemma)
- **Global:** Special tokens (Longformer)
- **Random:** Random patterns (BigBird)

😞 Low Adaptivity

😞 Low Flexibility

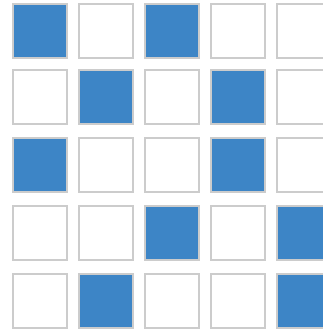
😄 Full Differentiability



$O(n \times k)$ time cost

$O(n \times k)$ memory cost

Dynamic Selection



- **Top-k:** Highest scores (NSA, MoBA, ...)
- **LSH:** Hash similarity (Reformer)
- **Cluster:** Group similarity (Routing)

😄 High Adaptivity

😄 More Flexibility

😞 Approx or No Differentiability



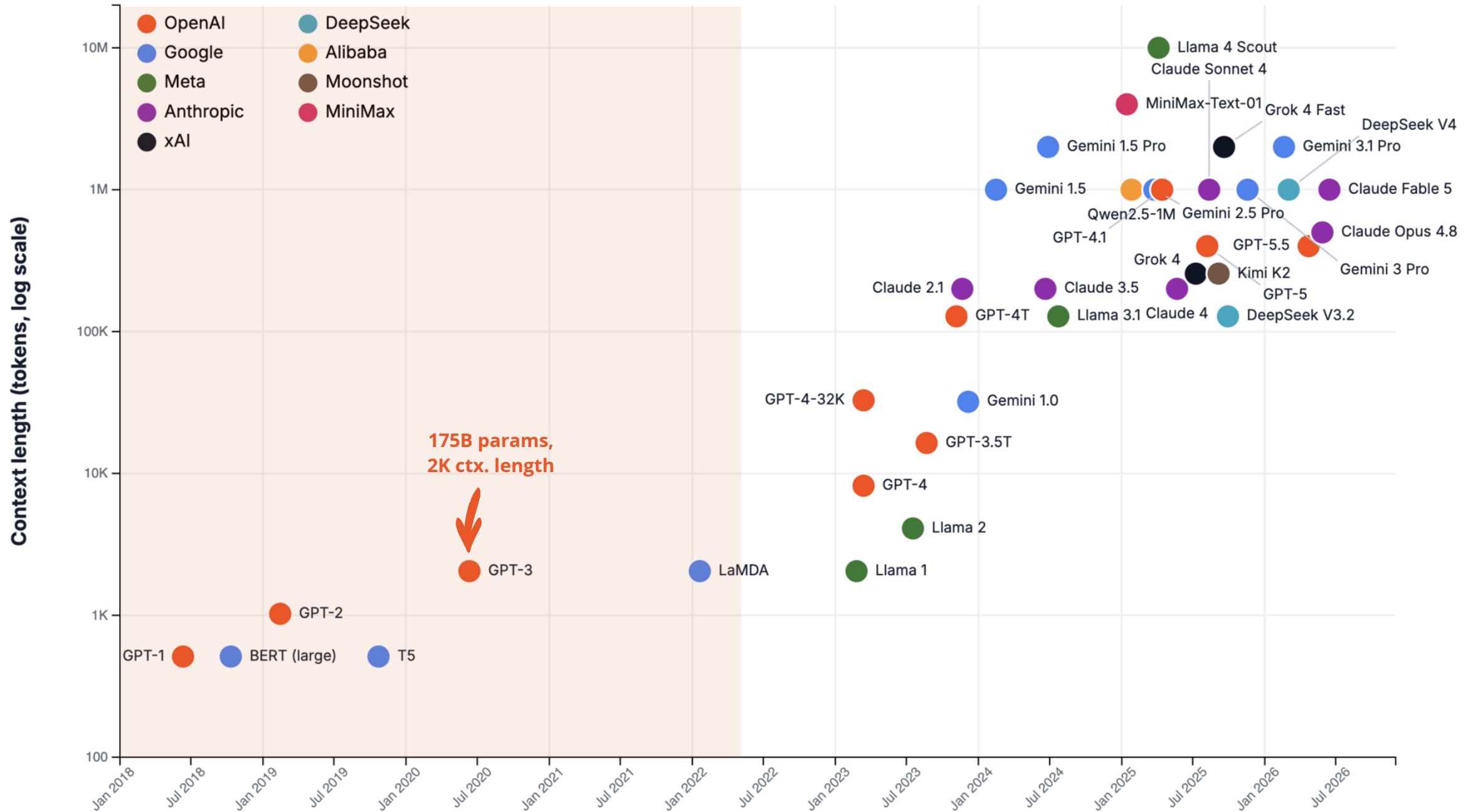
$O(n \log n)$ time cost

$O(n \log n)$ memory cost

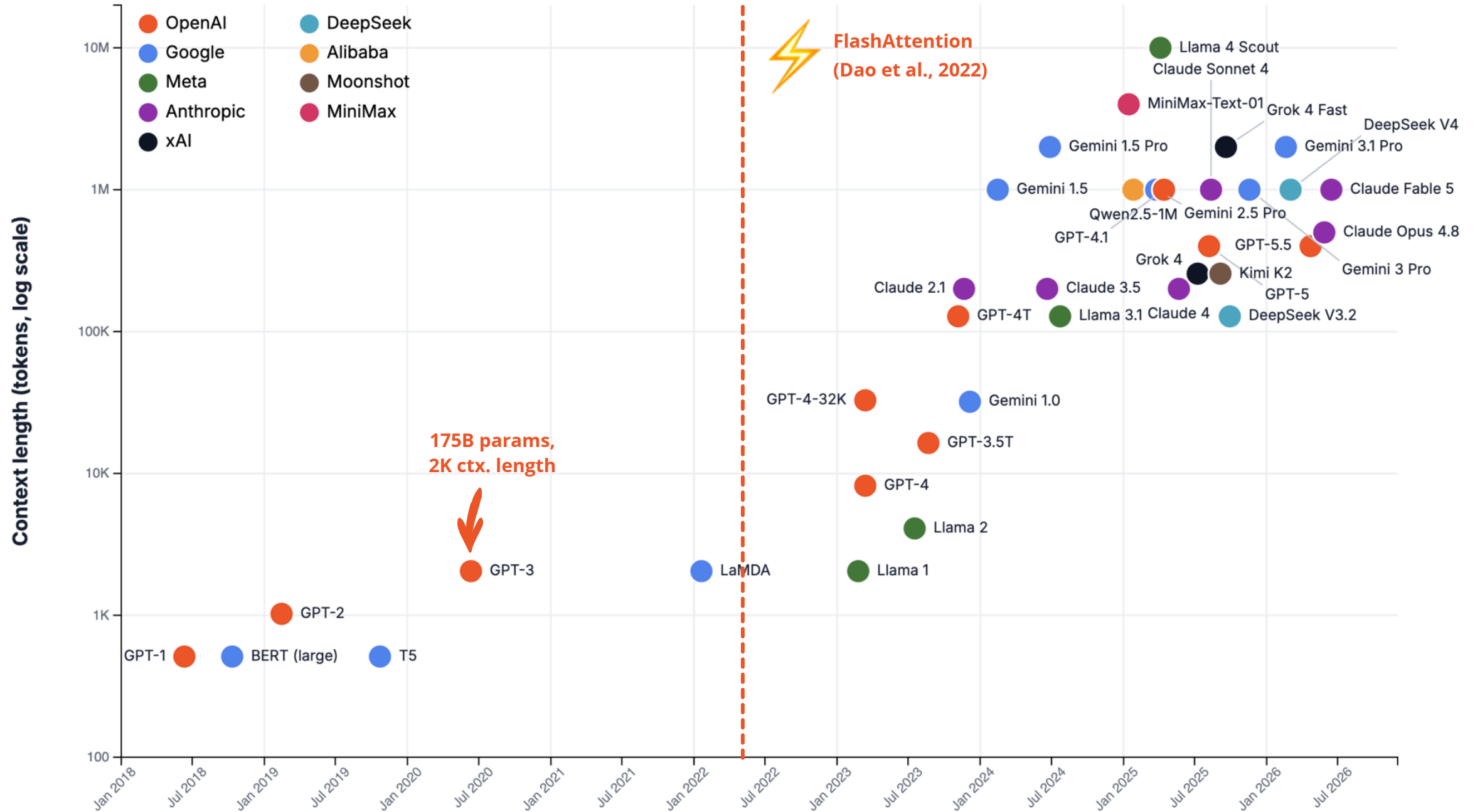
💥 Not used in practice...

Accuracy traded for efficiency!

LLM Context Length Evolution (2018–2026)

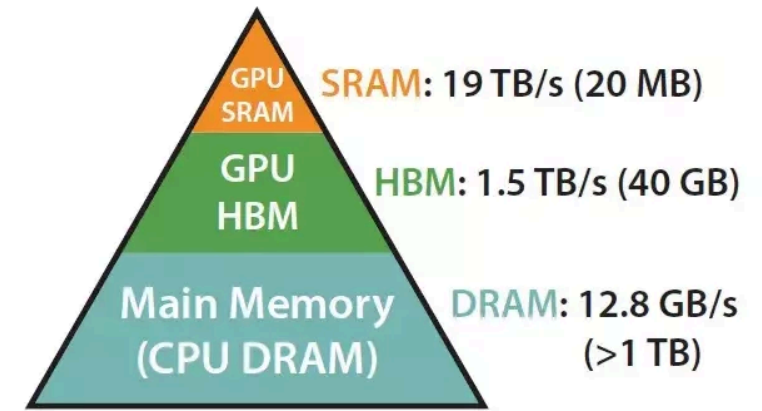


LLM Context Length Evolution (2018–2026)



⚡ Flash Attention

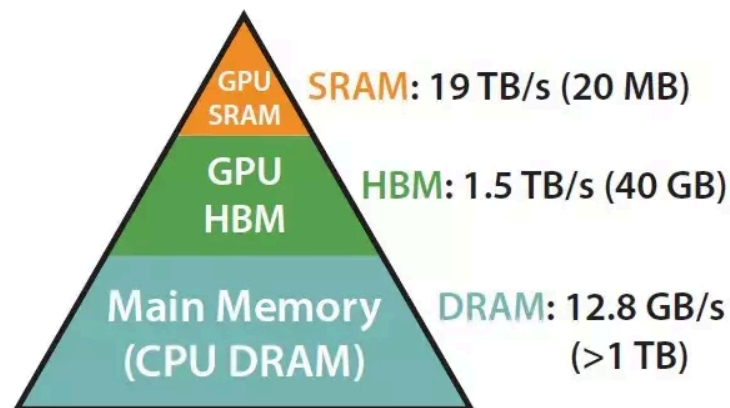
Hardware-aware implementation of softmax attention



Memory Hierarchy with
Bandwidth & Memory Size

⚡ Flash Attention

Hardware-aware implementation of softmax attention



Memory Hierarchy with
Bandwidth & Memory Size

```
1 Z = torch.matmul(Q, K.transpose(-1, -2))
2 P = torch.softmax(Z, dim=-1)
3 O = torch.matmul(P, V)
```

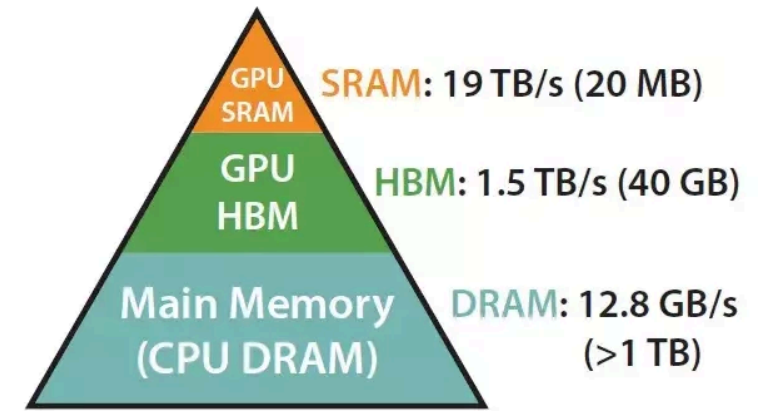
1. Load Q and K from HBM → compute matmul in SRAM → store Z in HBM
2. Load Z from HBM → compute softmax in SRAM → store P in HBM
3. Load P and V from HBM → compute matmul in SRAM → store O in HBM

⚡ Flash Attention

Hardware-aware implementation of softmax attention

```
1 Z = torch.matmul(Q, K.transpose(-1, -2))  
2 P = torch.softmax(Z, dim=-1)  
3 O = torch.matmul(P, V)
```

1. Load Q and K from HBM → compute matmul in SRAM → store Z in HBM
2. Load Z from HBM → compute softmax in SRAM → store P in HBM
3. Load P and V from HBM → compute matmul in SRAM → store O in HBM



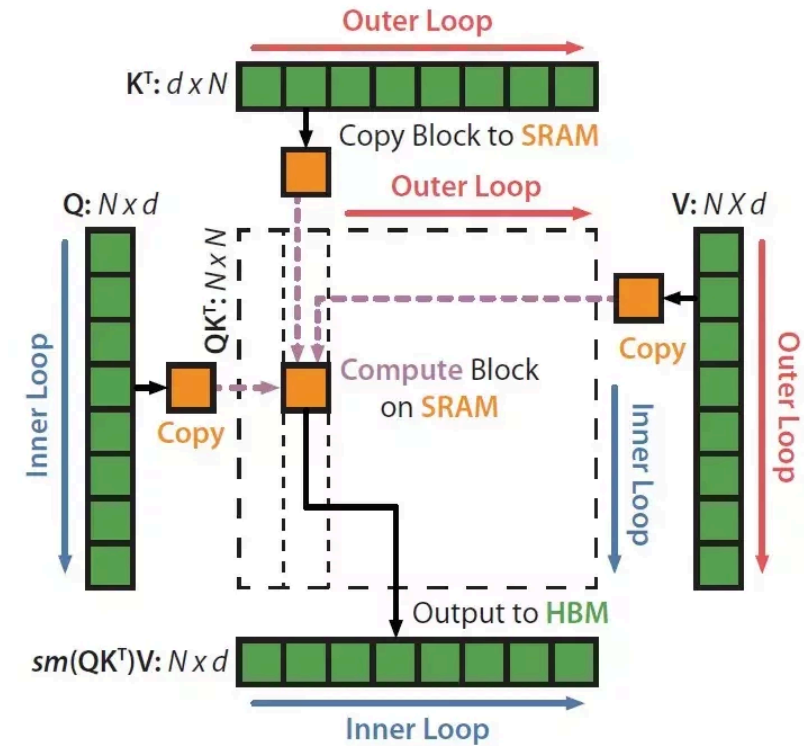
Memory Hierarchy with Bandwidth & Memory Size

```
1 O = flash_attn(Q, K, V)
```

Load Q, K and V from HBM,
Compute Z and P in SRAM,
Store O in HBM

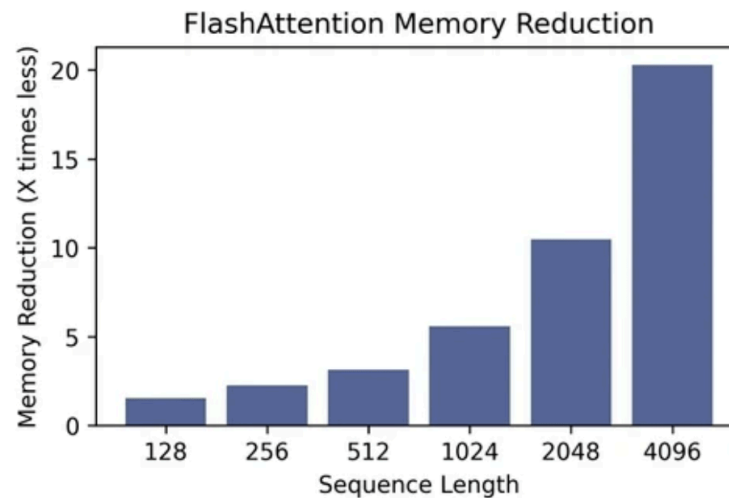
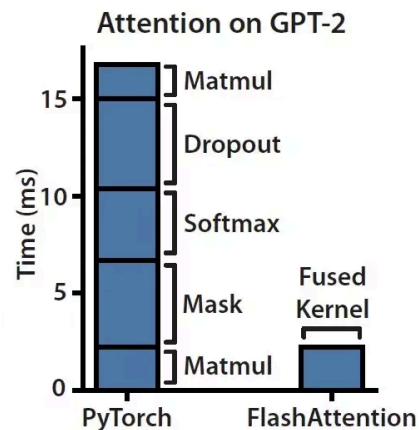
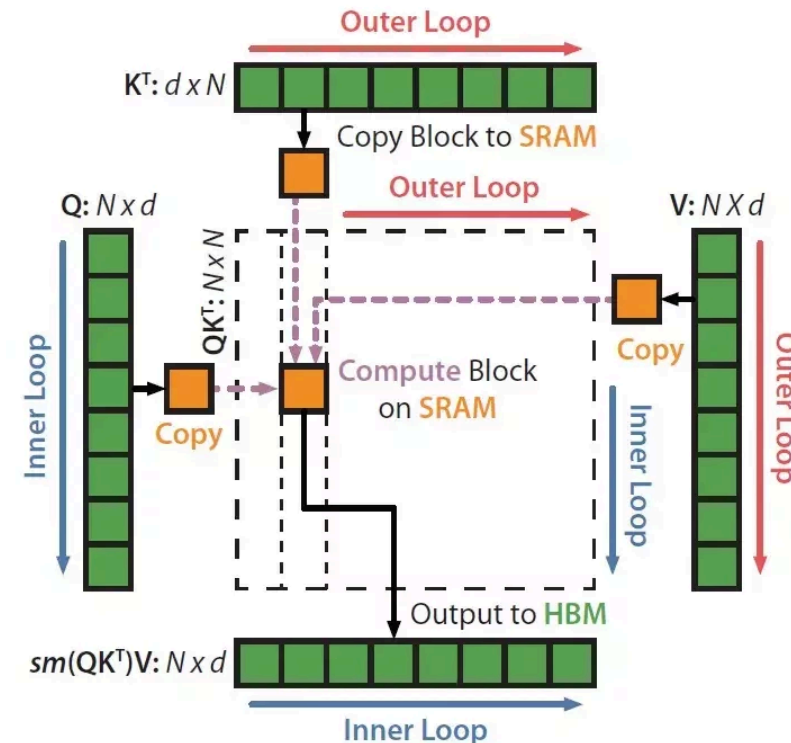
⚡ Flash Attention

- **Tiling:** work with blocks rather than single vectors
- **Online softmax:** more FLOPs but more efficient while also being exact!
- **Recomputation:** how to backprop through this?
-> recompute attention in the backward pass



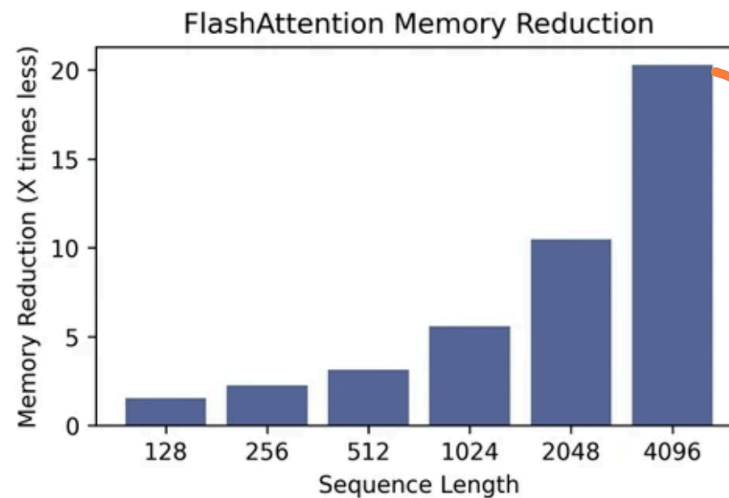
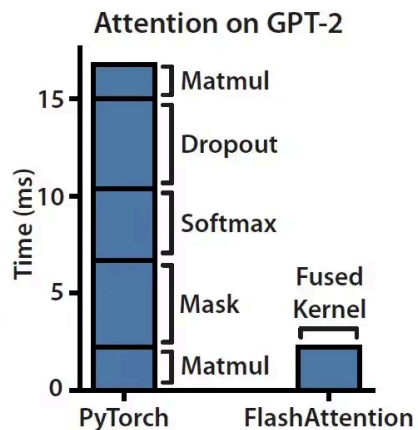
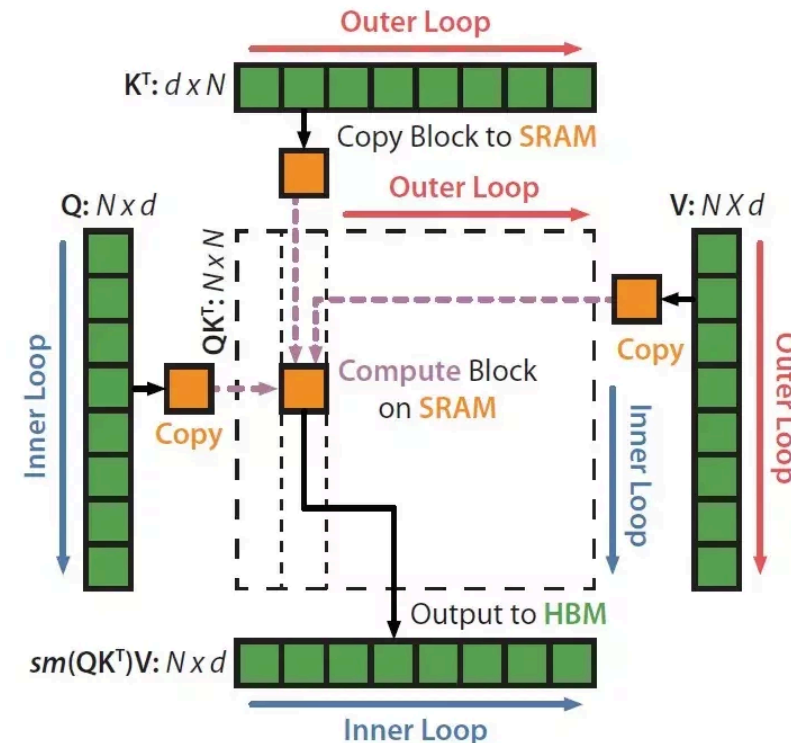
⚡ Flash Attention

- **Tiling:** work with blocks rather than single vectors
- **Online softmax:** more FLOPs but more efficient while also being exact!
- **Recomputation:** how to backprop through this?
 - > recompute attention in the backward pass



⚡ Flash Attention

- **Tiling:** work with blocks rather than single vectors
- **Online softmax:** more FLOPs but more efficient while also being exact!
- **Recomputation:** how to backprop through this?
 - > recompute attention in the backward pass



~20x reduction!

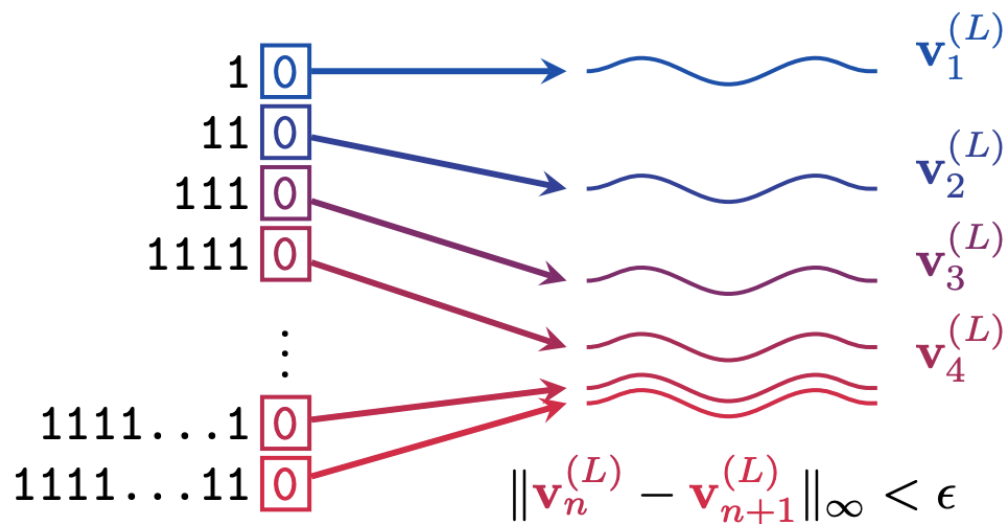
The Sweet Lesson

Can we just keep increasing the context length?

The Sweet Lesson

Can we just keep increasing the context length?

Representational Collapse

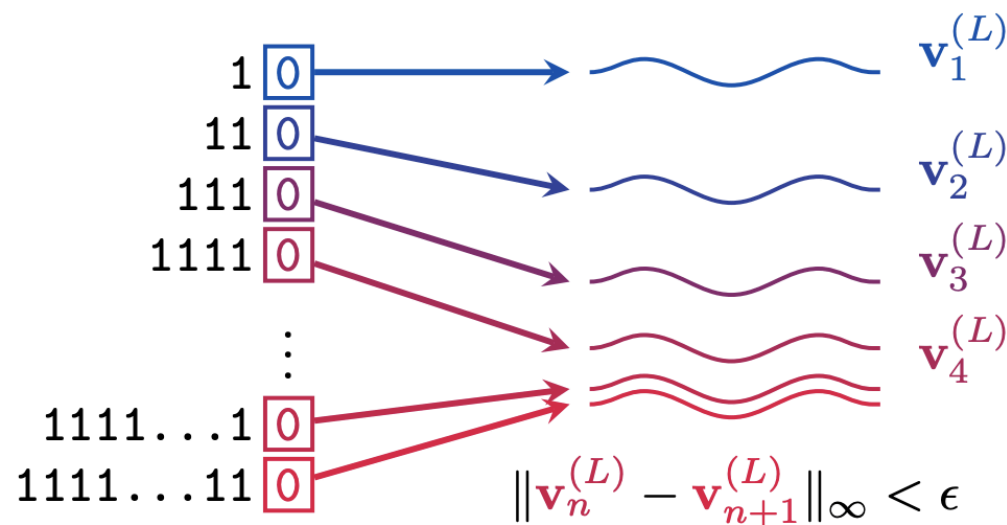


As we add context tokens, the final representation of "0" collapse to a single vector

The Sweet Lesson

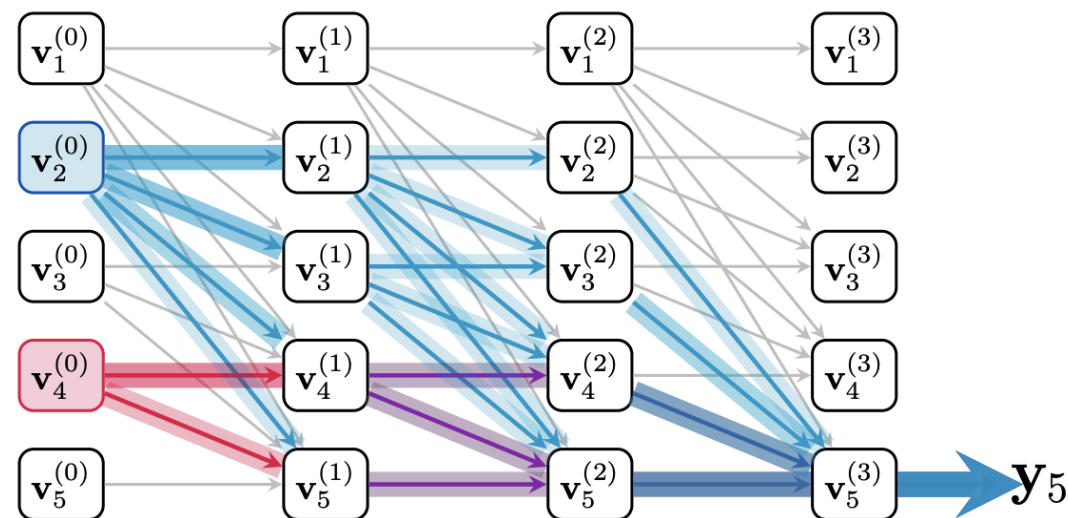
Can we just keep increasing the context length?

Representational Collapse



As we add context tokens, the final representation of "0" collapse to a single vector

Oversquashing

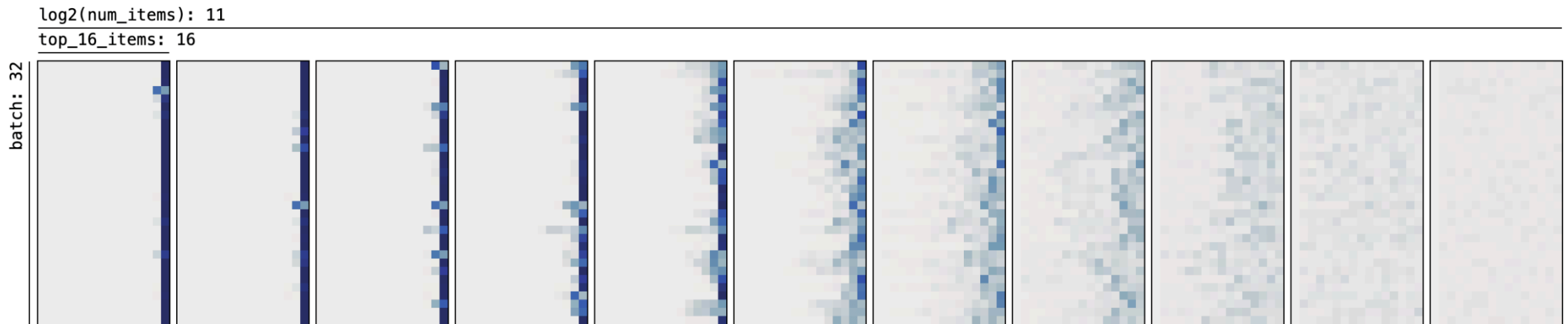


A lot of information from context tokens is forced through a single, fixed-size bottleneck

The Sweet Lesson

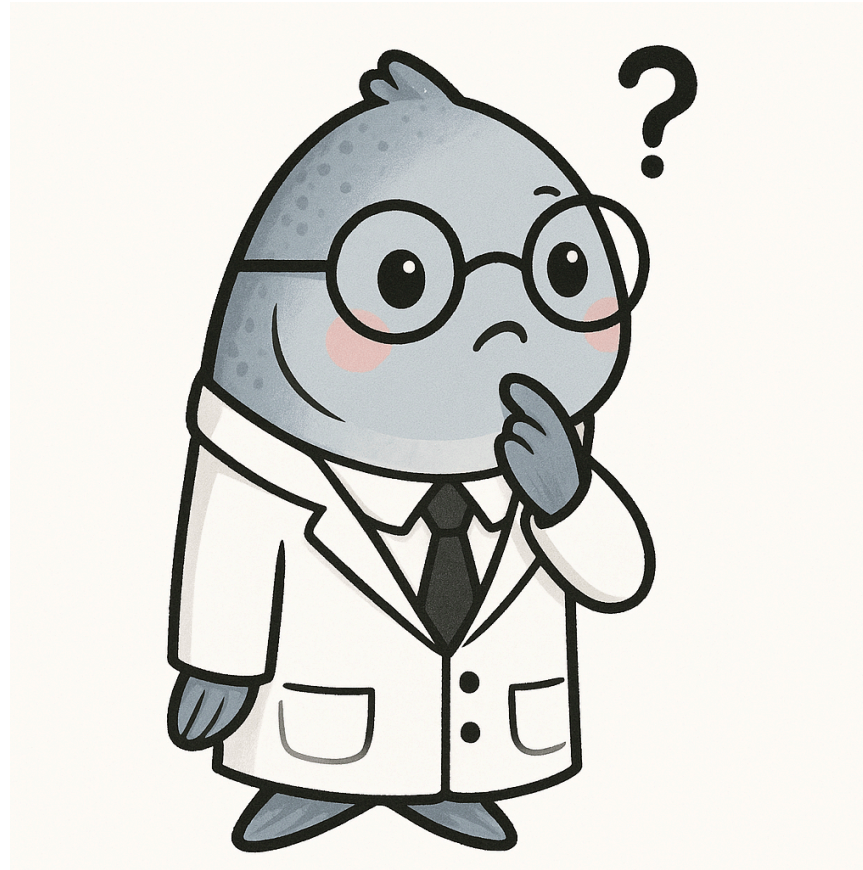
Can we just keep increasing the context length?

Attention Dispersion



As the number of tokens increase, the attention coefficients spread out
=> uniform distribution (simple average)

What is going on?

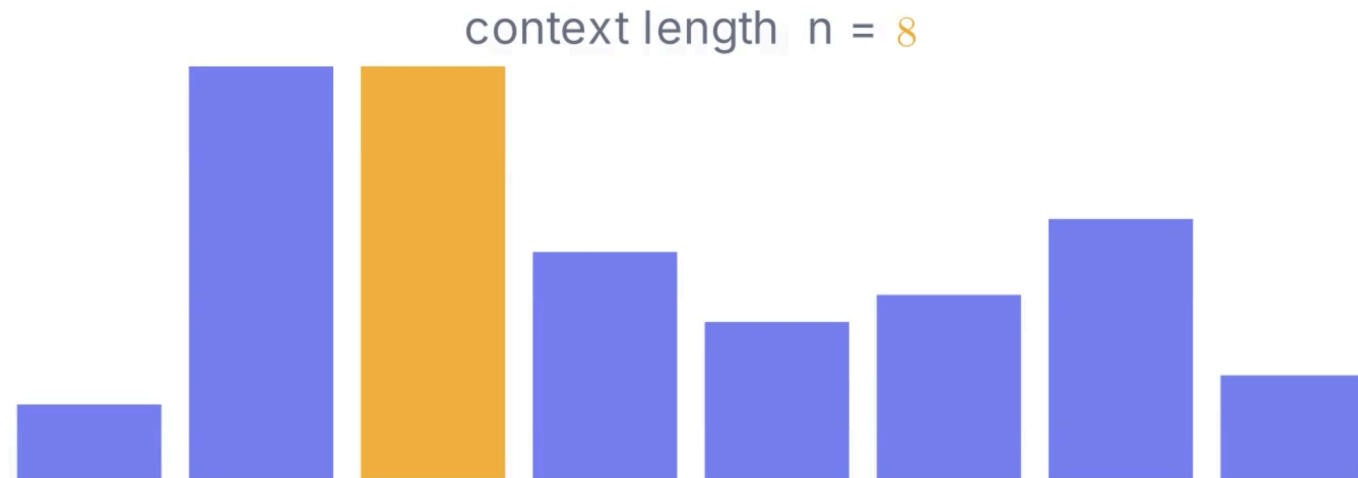


What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse

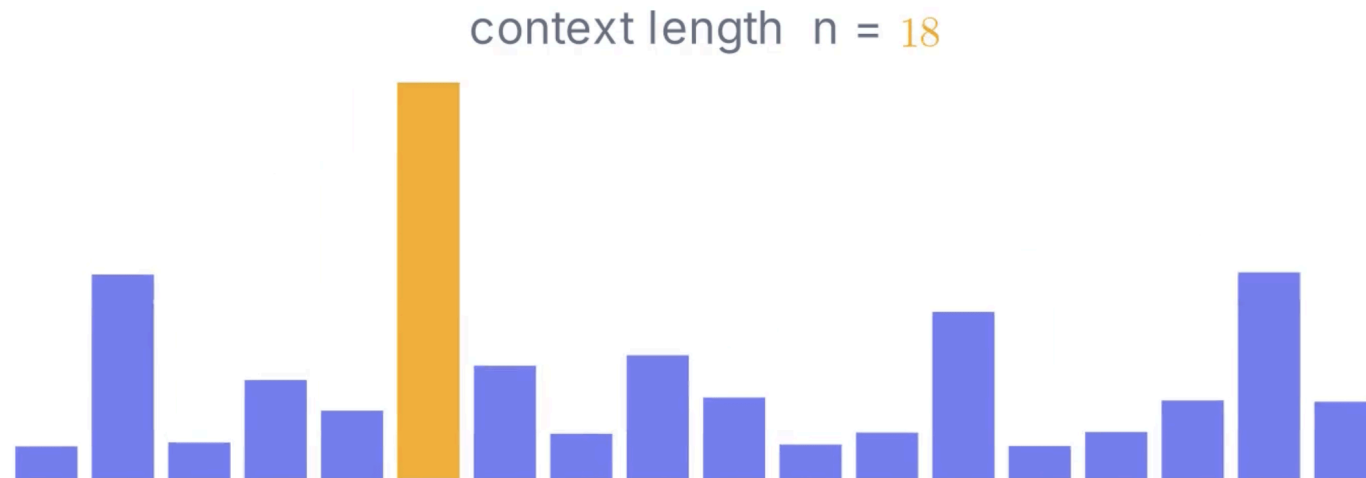


What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse



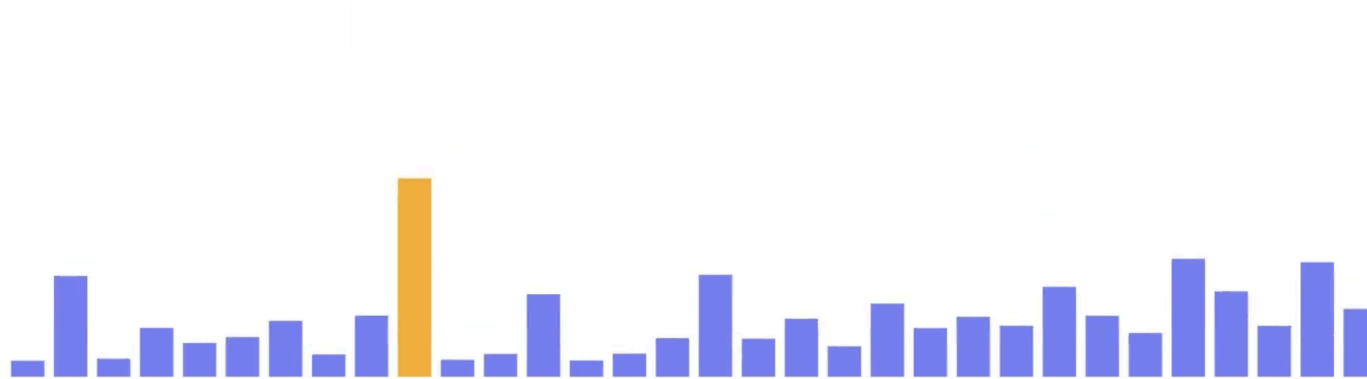
What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse

context length $n = 32$



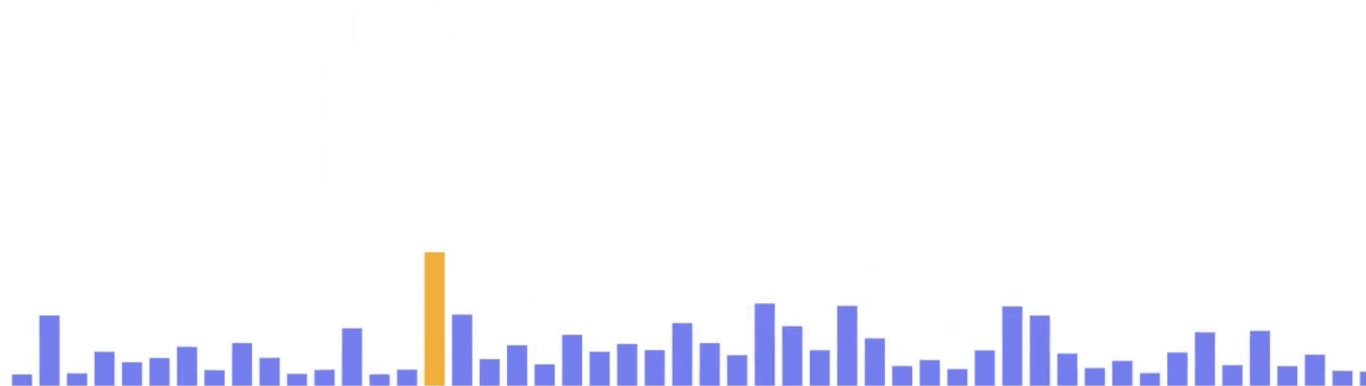
What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse

context length $n = 50$



What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse

context length $n = 106$



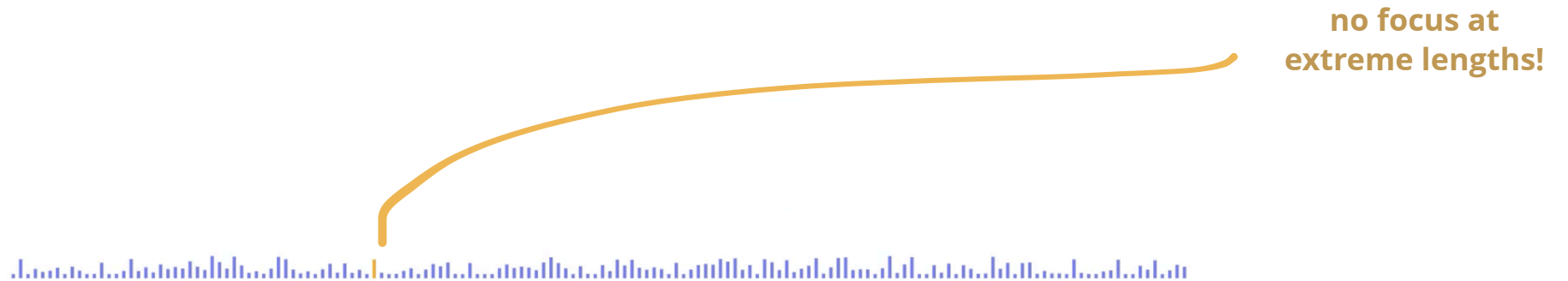
What is going on?

Long story short: the culprit is **softmax**!

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^n \exp(\mathbf{z}_k)}$$

- The exp function is always positive => dispersion
- All "computation paths" are active => oversquashing
- All "tokens" contribute => representational collapse

context length $n = 160$

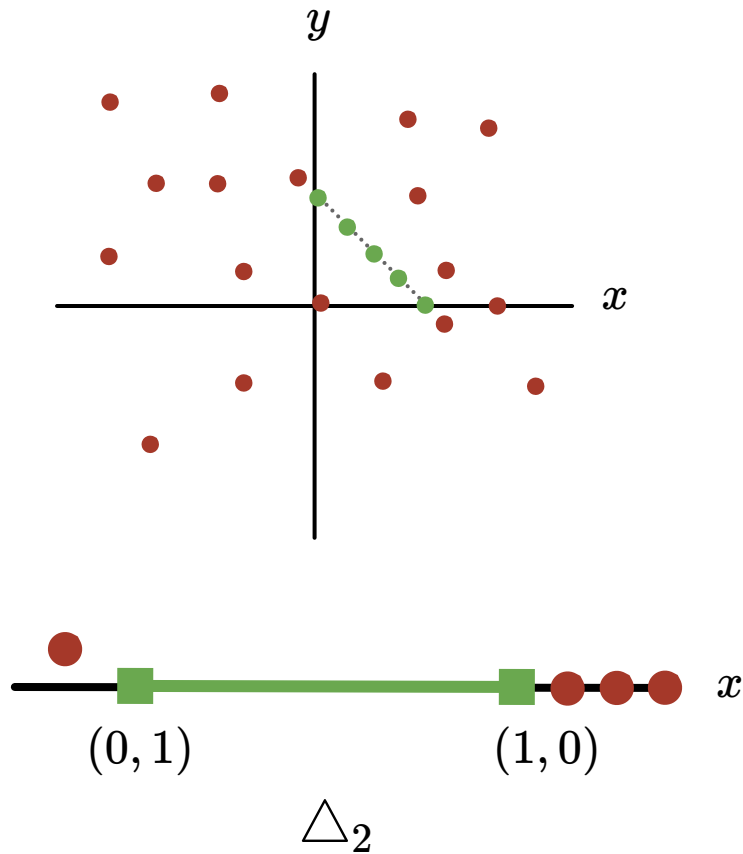


Softmax origins: probability simplex

$$\triangle_n = \{\mathbf{p} \in \mathbb{R}_+^n \mid \sum_i p_i = 1\}$$

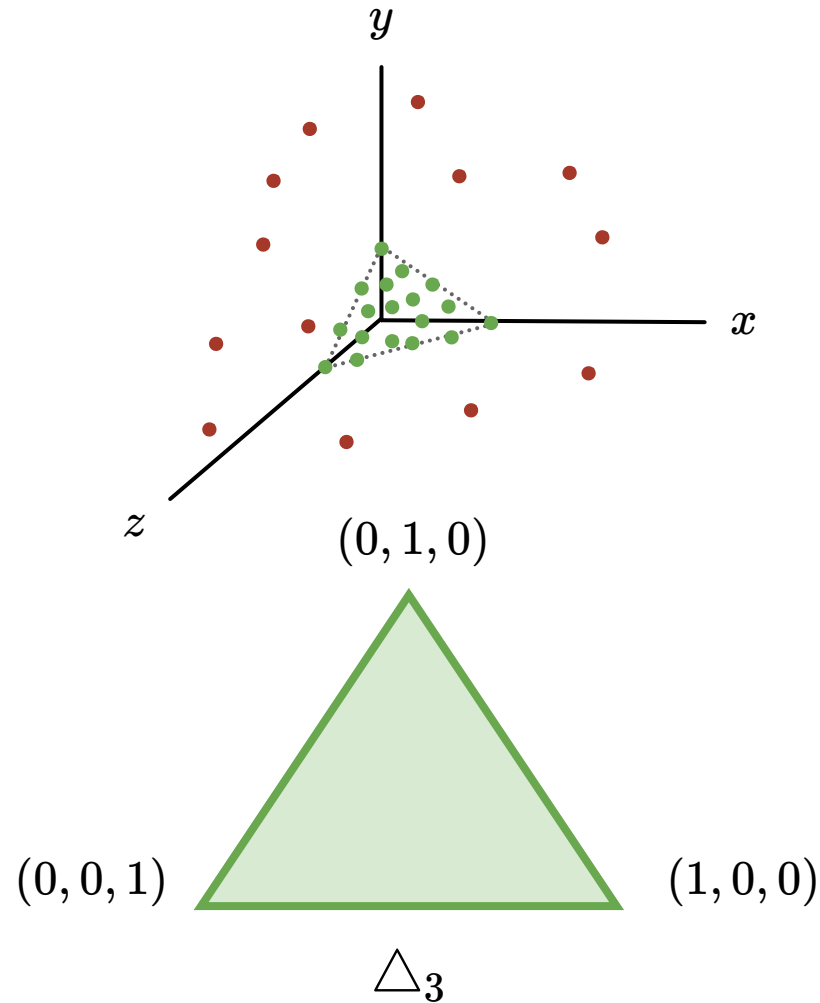
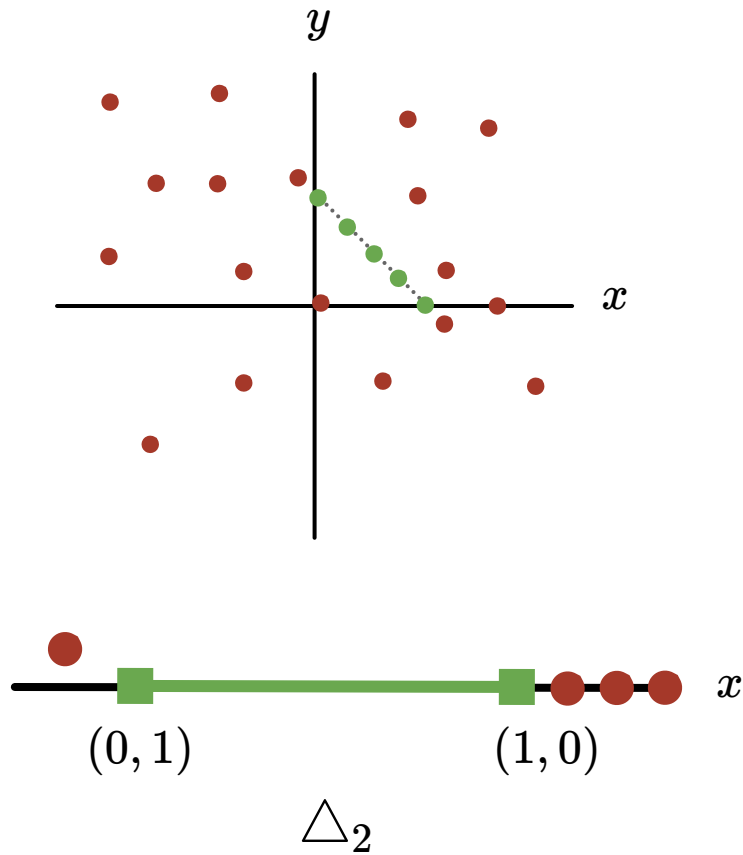
Softmax origins: probability simplex

$$\Delta_n = \{\mathbf{p} \in \mathbb{R}_+^n \mid \sum_i p_i = 1\}$$



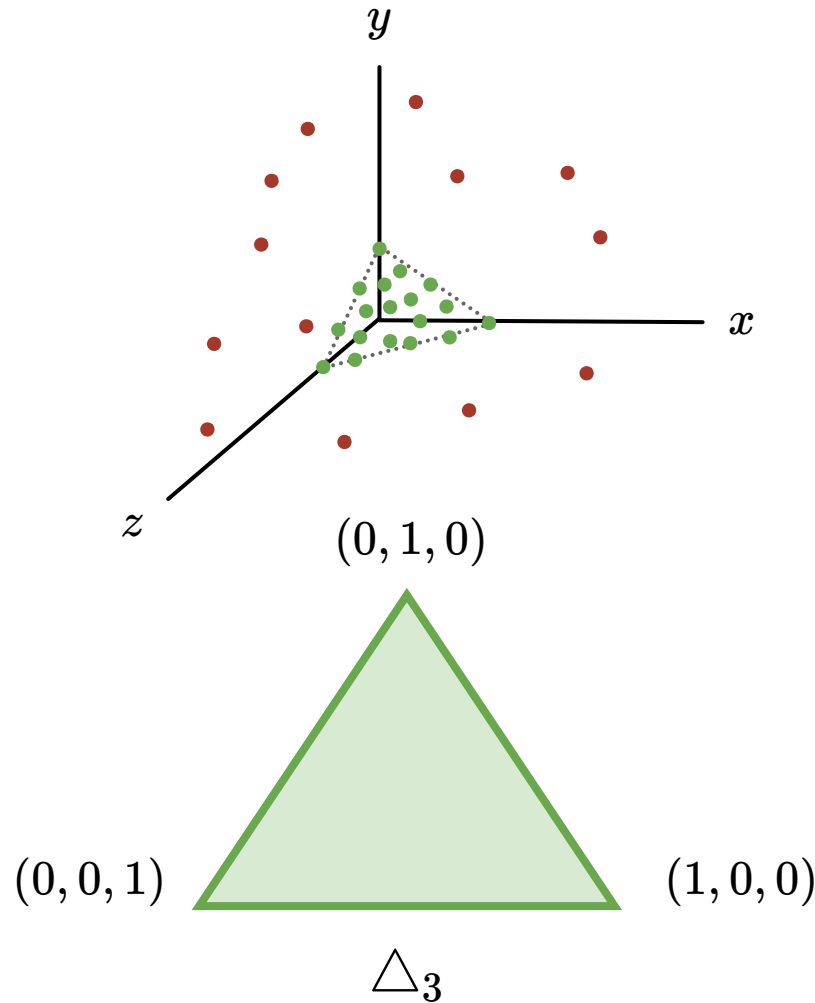
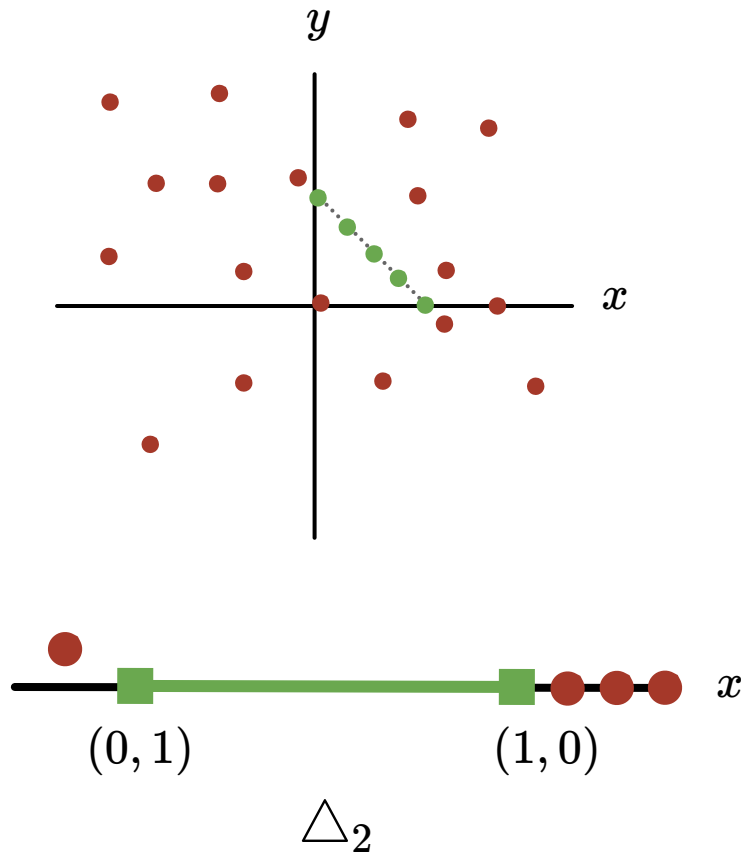
Softmax origins: probability simplex

$$\Delta_n = \{\mathbf{p} \in \mathbb{R}_+^n \mid \sum_i p_i = 1\}$$

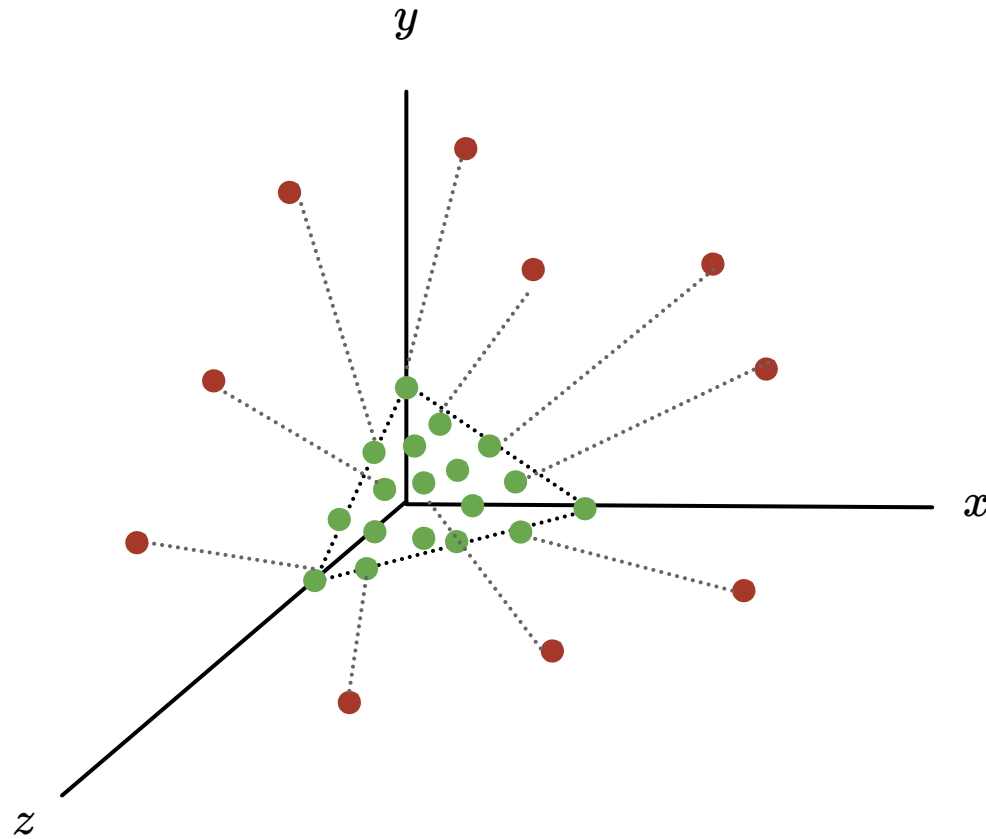


Softmax origins: probability simplex

$$\Delta_n = \{\mathbf{p} \in \mathbb{R}_+^n \mid \sum_i p_i = 1\}$$

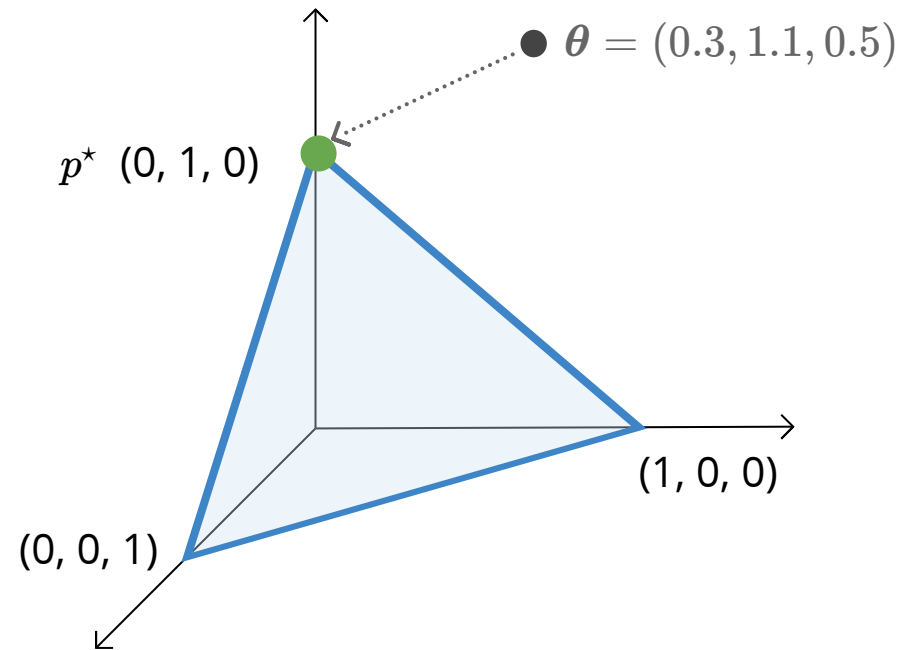


How can we map a score to a probability?



How can we map a score to a probability?

$$\mathbf{p}^* = \arg \max_{\mathbf{p} \in \Delta_n} \mathbf{p}^\top \mathbf{z}$$

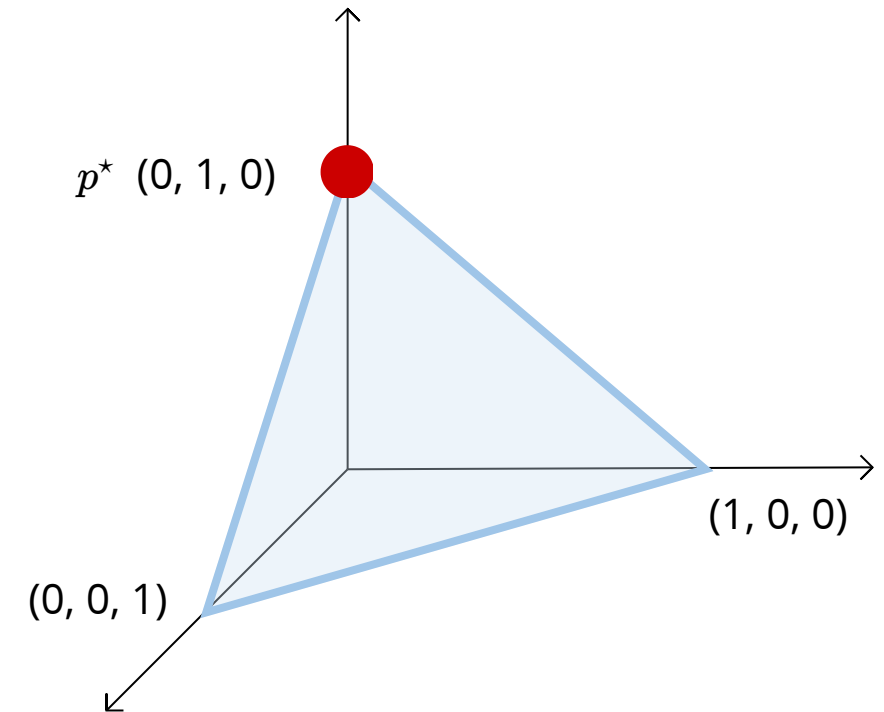


How can we map a score to a probability?

$$\mathbf{p}^* = \arg \max_{\mathbf{p} \in \Delta_n} \mathbf{p}^\top \mathbf{z} - \Omega(\mathbf{p})$$

↪ Entropy regularizer

● argmax: $\Omega(\mathbf{p}) = 0$

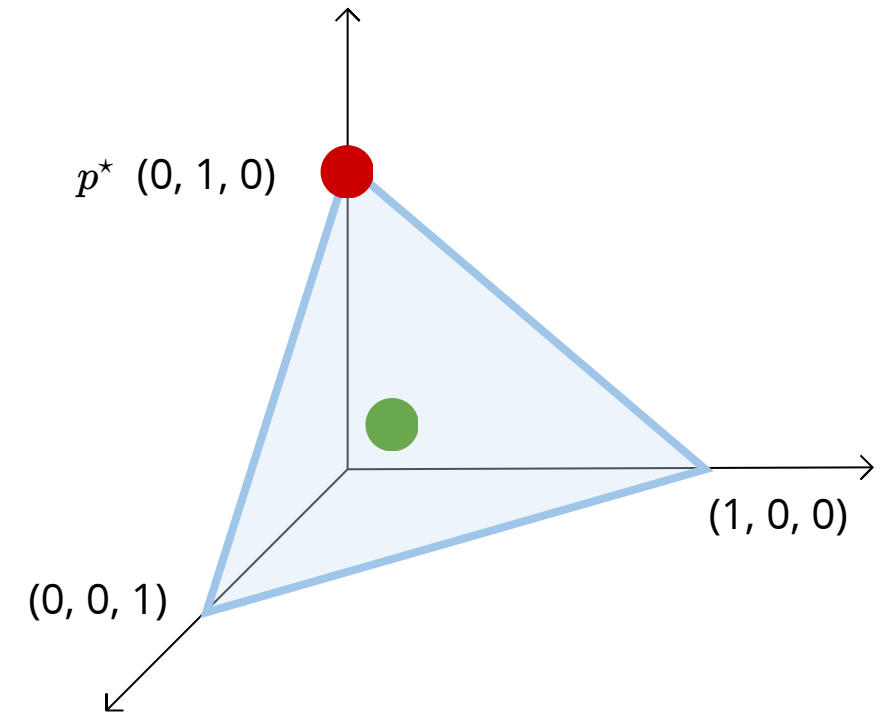


How can we map a score to a probability?

$$\mathbf{p}^* = \arg \max_{\mathbf{p} \in \Delta_n} \mathbf{p}^\top \mathbf{z} - \Omega(\mathbf{p})$$

↗ Entropy regularizer

- argmax: $\Omega(\mathbf{p}) = 0$
- softmax: $\Omega(\mathbf{p}) = \sum_j p_j \log p_j$ Neg. Shannon entropy

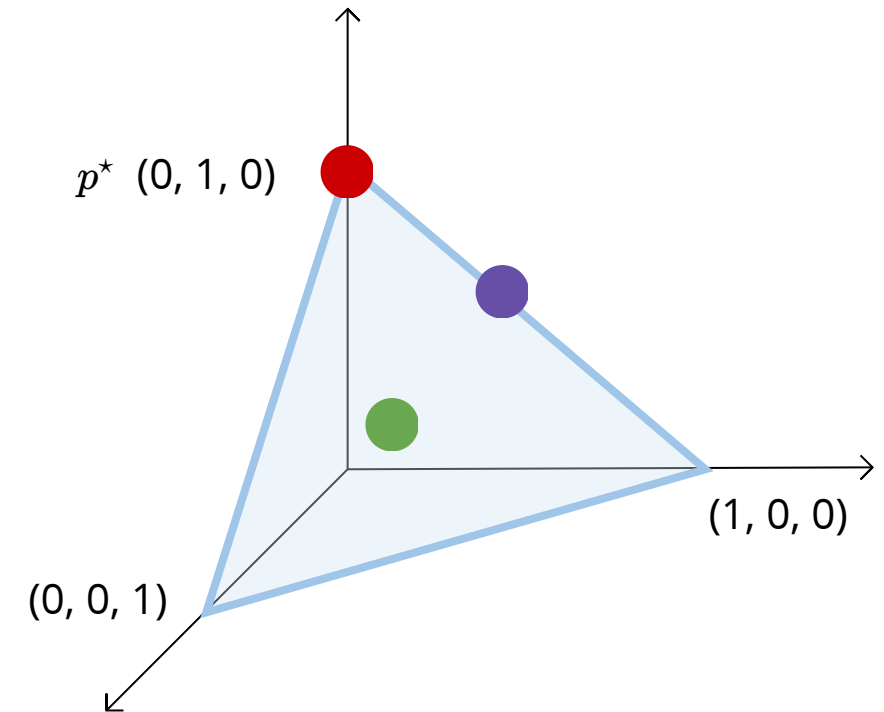


How can we map a score to a probability?

$$\mathbf{p}^* = \arg \max_{\mathbf{p} \in \Delta_n} \mathbf{p}^\top \mathbf{z} - \Omega(\mathbf{p})$$

↗ Entropy regularizer

- argmax: $\Omega(\mathbf{p}) = 0$
- softmax: $\Omega(\mathbf{p}) = \sum_j p_j \log p_j$ Neg. Shannon entropy
- sparsemax: $\Omega(\mathbf{p}) = \frac{1}{2} \|\mathbf{p}\|_2^2$ ~ Gini entropy

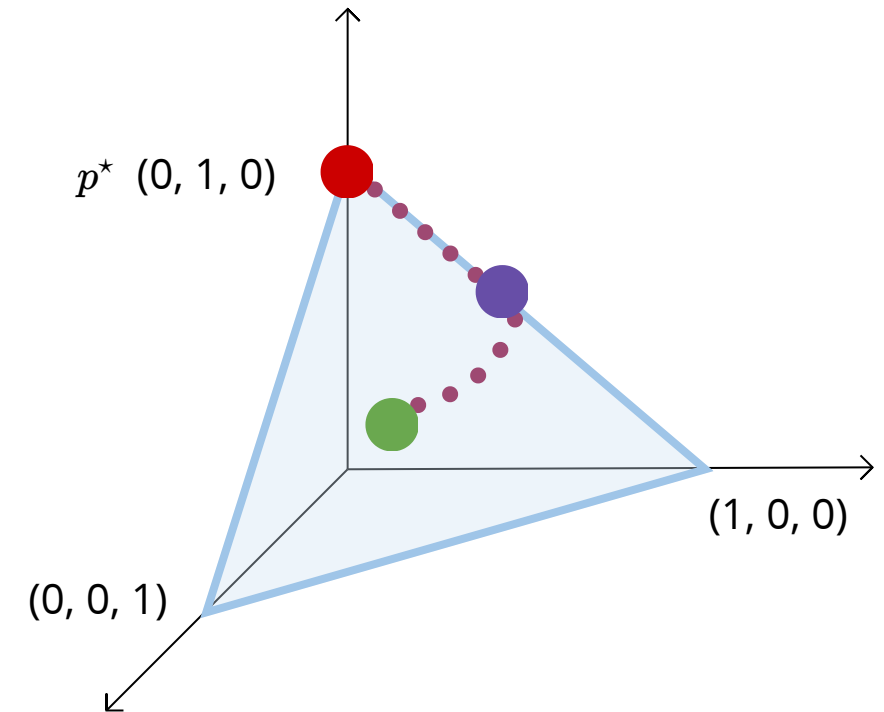


How can we map a score to a probability?

$$\mathbf{p}^* = \arg \max_{\mathbf{p} \in \Delta_n} \mathbf{p}^\top \mathbf{z} - \Omega(\mathbf{p})$$

↗ Entropy regularizer

- argmax: $\Omega(\mathbf{p}) = 0$
- softmax: $\Omega(\mathbf{p}) = \sum_j p_j \log p_j$ Neg. Shannon entropy
- sparsemax: $\Omega(\mathbf{p}) = \frac{1}{2} \|\mathbf{p}\|_2^2$ ~ Gini entropy
- α -entmax: $\Omega(\mathbf{p}) = \frac{1}{\alpha(\alpha-1)} \sum_j p_j^\alpha$ Tsallis entropy
 - softmax when $\alpha \rightarrow 1$
 - sparsemax when $\alpha = 2$



α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

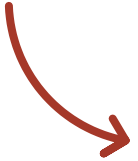
$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

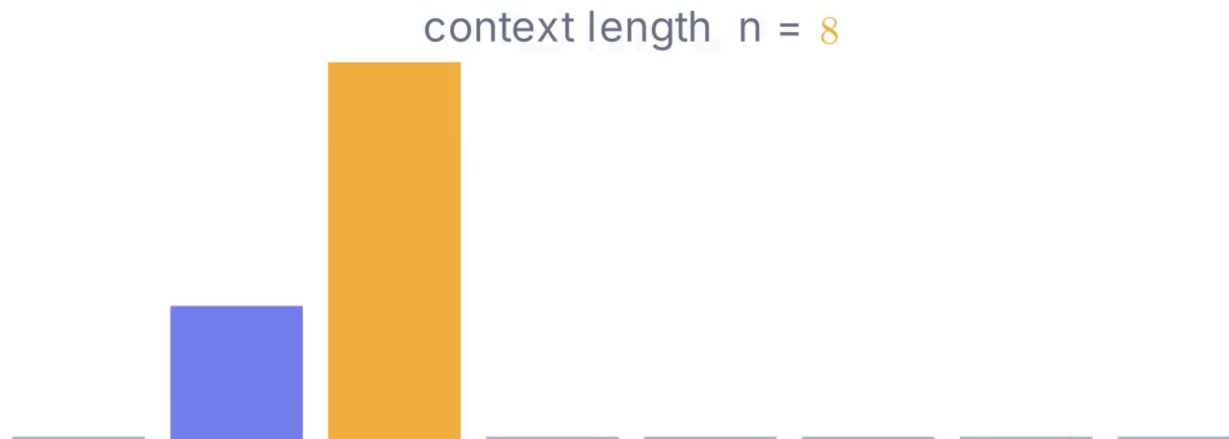

such that $\sum_i p_i = 1$

α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

 such that $\sum_i p_i = 1$

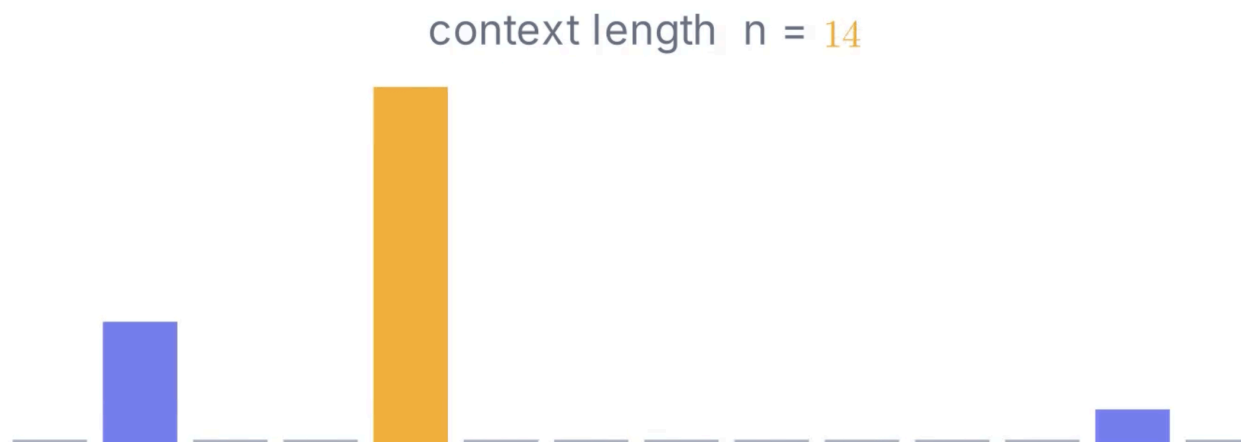


α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

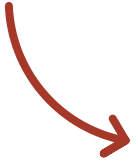
 such that $\sum_i p_i = 1$



α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

 such that $\sum_i p_i = 1$

context length $n = 41$



α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

 such that $\sum_i p_i = 1$

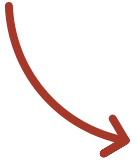
context length $n = 86$



α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

 such that $\sum_i p_i = 1$

context length $n = 116$



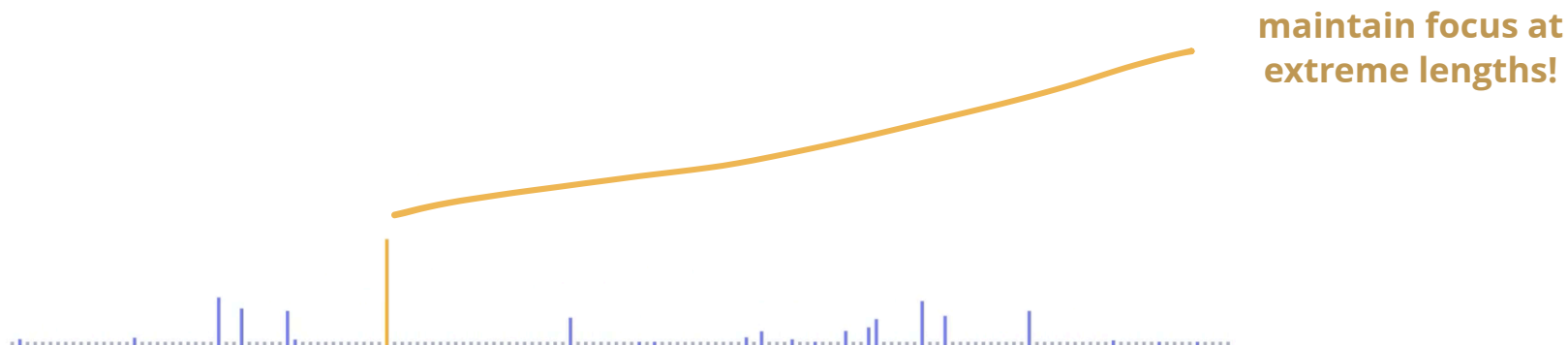
α -entmax

The output of α -entmax($\mathbf{z}$) is quite simple:

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}} \xrightarrow{\text{for } \alpha = 2} \mathbf{p} = \max(0, \mathbf{z} - \tau)$$

 such that $\sum_i p_i = 1$

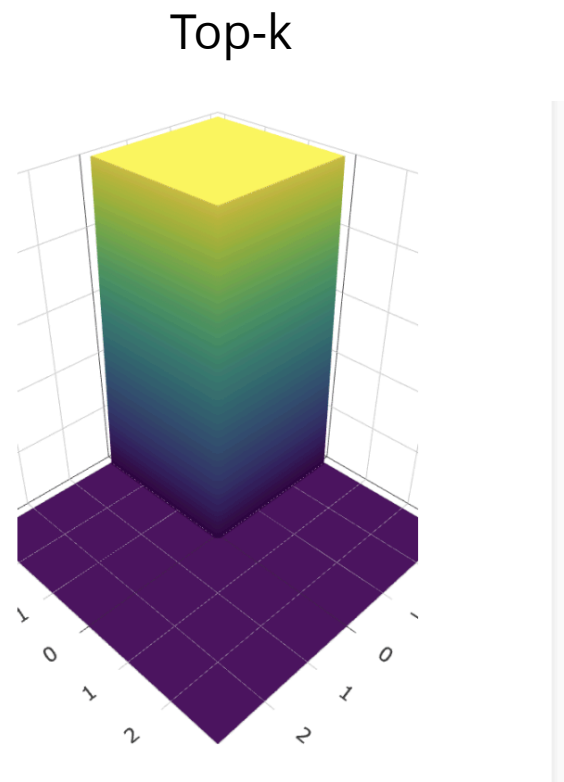
context length $n = 160$



α -entmax



But, why complicate so much? If we want sparsity, why not using top-k or "normalized ReLU"?



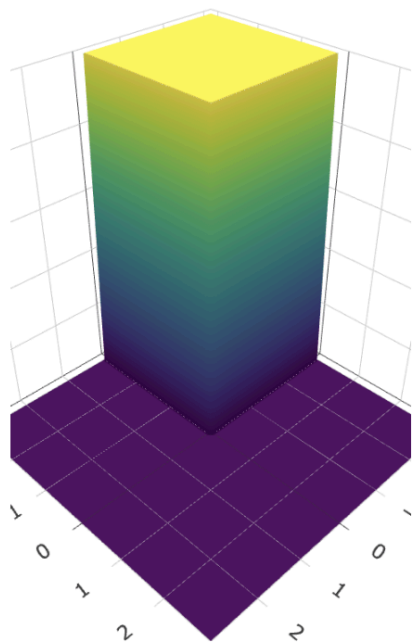
Slow to evaluate

Not GPU-friendly

Non-differentiable

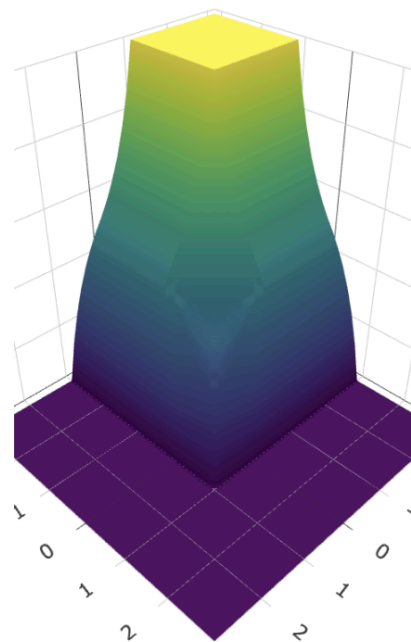
Controllable

Top-k



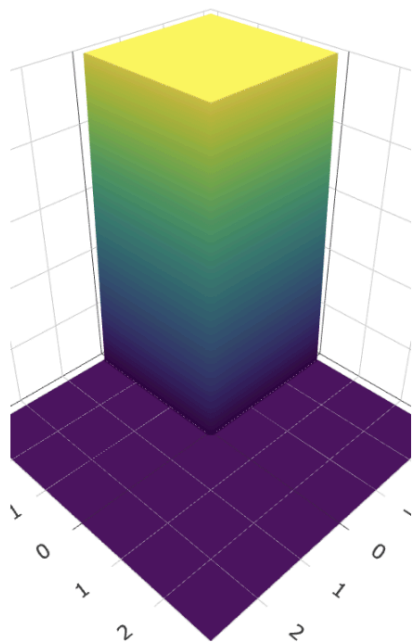
Slow to evaluate
Not GPU-friendly
Non-differentiable
Controllable

ReLU-norm

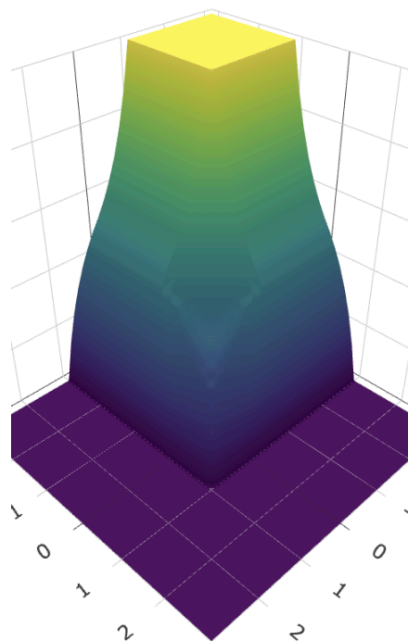


Easy to evaluate
GPU-friendly
Diff. but tricky (e.g. 0/0)
Not controllable

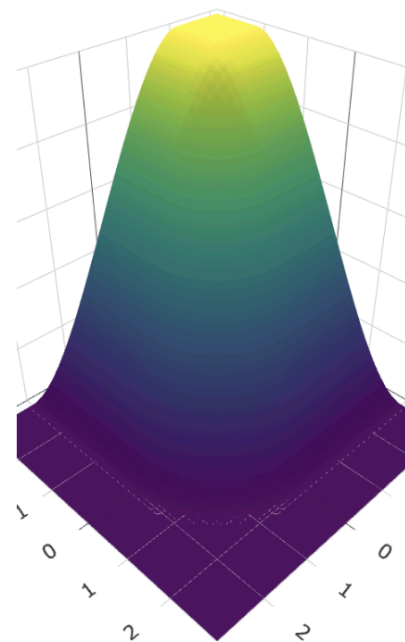
Top-k



ReLU-norm



1.5-entmax



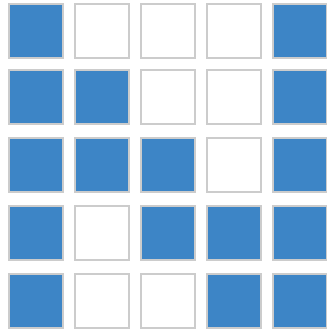
Slow to evaluate
Not GPU-friendly
Non-differentiable
Controllable

Easy to evaluate
GPU-friendly
Diff. but tricky (e.g. 0/0)
Not controllable

Easy to evaluate
GPU-friendly*
Differentiable
Controllable

Sparse Attention

Fixed Patterns



- **Window:** Local context (Longformer)
- **Global:** Special tokens (Longformer)
- **Random:** Random patterns (BigBird)

😞 Low Adaptivity

😞 Low Flexibility

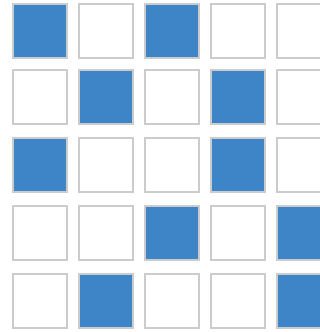
😞 Full Differentiability



$O(n \times k)$ time cost

$O(n \times k)$ memory cost

Dynamic Selection



- **Top-k:** Highest scores (NSA)
- **LSH:** Hash similarity (Reformer)
- **Cluster:** Group similarity (Routing)

😊 High Adaptivity

😐 Medium Flexibility

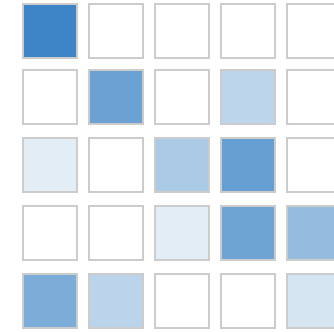
😞 Approx or No Differentiability



$O(n \log n)$ time cost

$O(n \log n)$ memory cost

Sparse Distribution



- **α -entmax:** natural sparse transformation
- α controls sparsity

😊 High Adaptivity

😊 High Flexibility

😊 Full Differentiability



$O(n^2 \log n)$ time cost

$O(n^2)$ memory cost

α -entmax

$$\mathbf{p} = \max(0, (\alpha - 1)\mathbf{z} - \tau)^{\frac{1}{\alpha-1}}$$

such that $\sum_i p_i = 1$

● $\alpha = 2$ (sparsemax)

Input: $\mathbf{z}$

Sort $\mathbf{z}$ as $z_{(1)} \geq \dots \geq z_{(K)}$

Find $k(\mathbf{z}) := \max \left\{ k \in [K] \mid 1 + kz_{(k)} > \sum_{j \leq k} z_{(j)} \right\}$

Define $\tau(\mathbf{z}) = \frac{(\sum_{j \leq k(\mathbf{z})} z_{(j)}) - 1}{k(\mathbf{z})}$

Output: $\mathbf{p}$ s.t. $p_i = [z_i - \tau(\mathbf{z})]_+$.

(Martins and Astudillo, 2016)

● $\alpha = 1.5$

Sort $\mathbf{z}$, yielding $z_{[d]} \leq \dots \leq z_{[1]}$; set $\mathbf{z} \leftarrow \mathbf{z}/2$

for $\rho \in 1, \dots, d$ **do**

$M(\rho) \leftarrow 1/\rho \sum_{j=1}^{\rho} z_{[j]}$

$S(\rho) \leftarrow \sum_{j=1}^{\rho} (z_{[j]} - M(\rho))^2$

$\tau(\rho) \leftarrow M(\rho) - \sqrt{1/\rho (1 - S(\rho))}$

if $z_{[\rho+1]} \leq \tau(\rho) \leq z_{[\rho]}$ **then**

return $\mathbf{p}^* = [\mathbf{z} - \tau \mathbf{1}]_+^2$

(Peters, Niculae, and Martins, 2019)

- Does not work for all values of α

- Requires sorting (not GPU-friendly)

Can we have a "Flash entmax attention"?

⚡ AdaSplash: Adapative Sparse Flash Attention

w/



Nuno Gonçalves



André F. T. Martins

 arxiv.org/abs/2502.12082

 github.com/deep-spin/adasplash

```
1 # pip install adasplash
2 from adasplash import adasplash
3
4 y = adasplash(q, k, v, alpha=1.5, niter=3, is_causal=True)
```


α -entmax Flash Attention

Algorithm 1 α -entmax Attention Kernel

Require: Fast shared memory of size M , sparsity control $\alpha \in (1, \infty]$.

Input: $Q \in \mathbb{R}^{N \times d}$, keys $K \in \mathbb{R}^{N \times d}$, and values $V \in \mathbb{R}^{N \times d}$.

- 1: Initialize $O = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\tau = (0)_N \in \mathbb{R}^N$ in slow memory.
 - 2: Divide Q, τ, O into T_r blocks of size $\lceil \frac{M}{4} \rceil$
 - 3: Divide K, V into T_c blocks of size $\lceil \frac{M}{4} \rceil$
 - 4: **for** $1 \leq i \leq T_r$ **do**
 - 5: Load Q_i, O_i, τ_i from slow to fast memory
 - 6: Compute $\tau_i(\alpha)$ on fast memory via Halley-bisection
 - 7: **for** $1 \leq j \leq T_c$ **do**
 - 8: Load K_j, V_j from slow to fast memory
 - 9: Compute $Z_{ij} = Q_i K_j^T$
 - 10: Compute $P_{ij} = \max(0, (\alpha - 1)Z_{ij} - \tau_i)^{1/(\alpha-1)}$
 - 11: Write $O_i = O_i + P_{ij} V_j$ to slow memory
 - 12: **end for**
 - 13: **end for**
 - 14: Return O .
-

α -entmax Flash Attention

Algorithm 1 α -entmax Attention Kernel

Require: Fast shared memory of size M , sparsity control $\alpha \in (1, \infty]$.

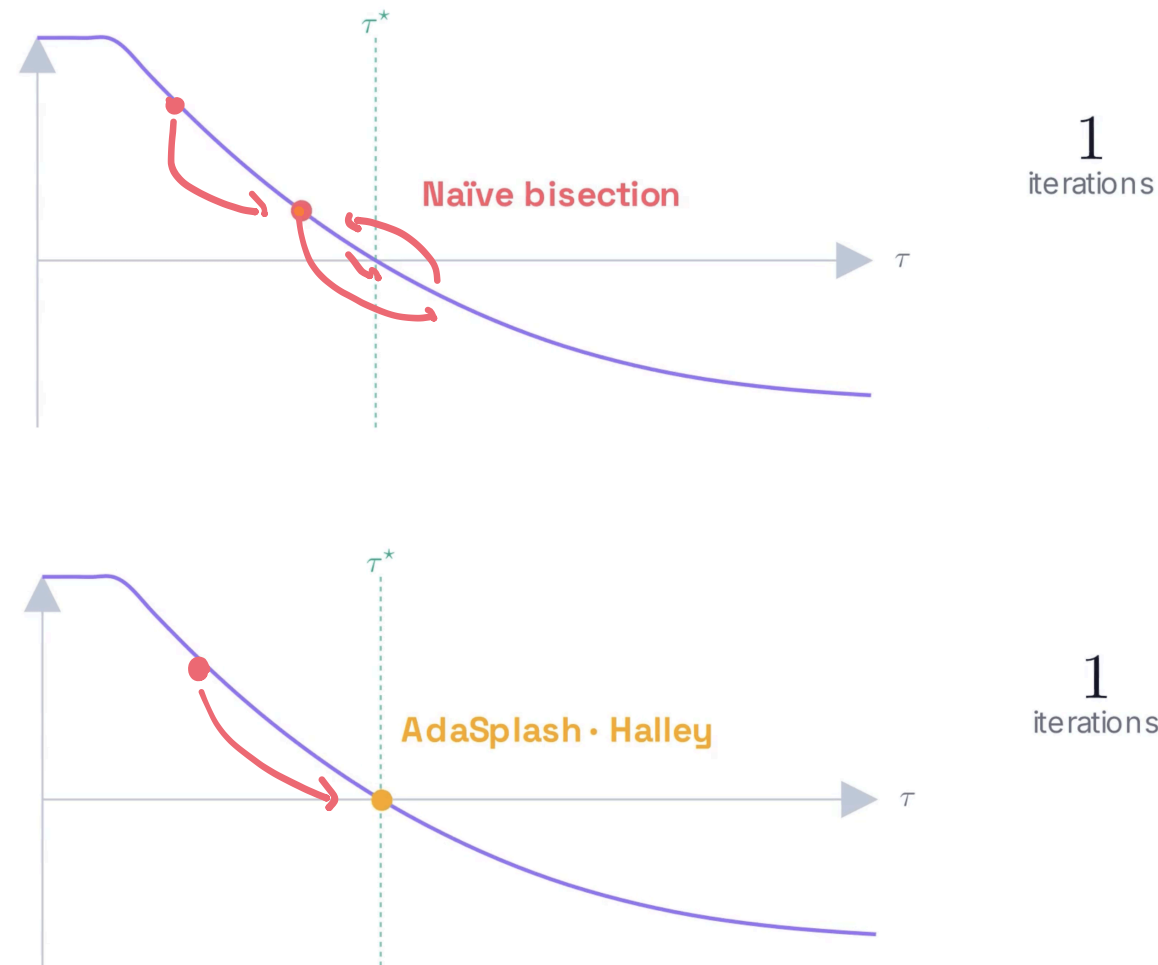
Input: $Q \in \mathbb{R}^{N \times d}$, keys $K \in \mathbb{R}^{N \times d}$, and values $V \in \mathbb{R}^{N \times d}$.

- 1: Initialize $O = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\tau = (0)_N \in \mathbb{R}^N$ in slow memory.
 - 2: Divide Q, τ, O into T_r blocks of size $\lceil \frac{M}{4} \rceil$
 - 3: Divide K, V into T_c blocks of size $\lceil \frac{M}{4} \rceil$
 - 4: **for** $1 \leq i \leq T_r$ **do**
 - 5: Load Q_i, O_i, τ_i from slow to fast memory
 - 6: Compute $\tau_i(\alpha)$ on fast memory via Halley-bisection
 - 7: **for** $1 \leq j \leq T_c$ **do**
 - 8: Load K_j, V_j from slow to fast memory
 - 9: Compute $Z_{ij} = Q_i K_j^T$
 - 10: Compute $P_{ij} = \max(0, (\alpha - 1)Z_{ij} - \tau_i)^{1/(\alpha-1)}$
 - 11: Write $O_i = O_i + P_{ij} V_j$ to slow memory
 - 12: **end for**
 - 13: **end for**
 - 14: Return O .
-

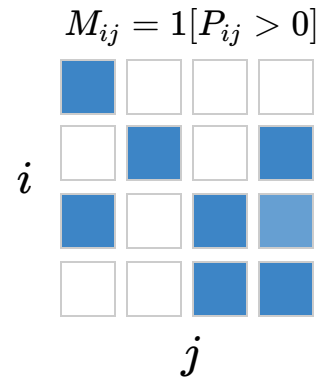
⚡ α -entmax Flash Attention

Algorithm 1 Halley-bisection algorithm for α -entmax.

- 1: **Input:** logits $\mathbf{s} \in \mathbb{R}^n$, param. $\alpha \in \mathbb{R}$, iterations T
 - 2: Define $f(\tau) := \sum_i [s_i - \tau]_+^{1/(\alpha-1)} - 1$
 - 3: Set $\mathbf{s} \leftarrow (\alpha - 1)\mathbf{s}$
 - 4: Initialize $\tau_{\text{lo}} = \max(\mathbf{s}) - 1$
 - 5: Initialize $\tau_{\text{hi}} = \max(\mathbf{s}) - n^{1-\alpha}$
 - 6: Initialize $\tau = (\tau_{\text{lo}} + \tau_{\text{hi}})/2$
 - 7: **repeat**
 - 8: Compute $\tau_{\text{lo}}, \tau_{\text{hi}} = B_f(\tau)$ (Equation 4)
 - 9: Compute $\tau_H = H_f(\tau)$ (Equation 5)
 - 10: **if** $\tau_H \in [\tau_{\text{lo}}, \tau_{\text{hi}}]$ **then**
 - 11: $\tau \leftarrow \tau_H$ ▷ (Halley's Update)
 - 12: **else**
 - 13: $\tau \leftarrow \frac{1}{2}(\tau_{\text{lo}} + \tau_{\text{hi}})$ ▷ (Bisection Update)
 - 14: **end if**
 - 15: **until** T iterations are completed
 - 16: **Output:** $[\mathbf{s} - \tau \mathbf{1}]_+^{1/(\alpha-1)}$
-



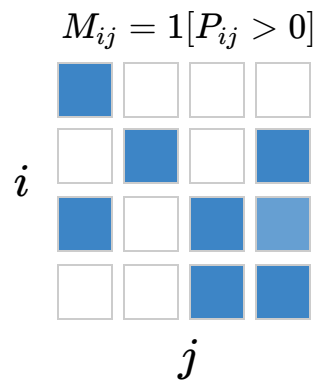
⚡ AdaSplash: Adapative Sparse Flash Attention



$$\mathcal{Q}_i = \{j \mid M_{ij} = 1\} \text{ (non-zero cols)}$$

$$\mathcal{K}_j = \{i \mid M_{ij} = 1\} \text{ (non-zero rows)}$$

⚡ AdaSplash: Adapative Sparse Flash Attention



$$\mathcal{Q}_i = \{j \mid M_{ij} = 1\} \text{ (non-zero cols)}$$

$$\mathcal{K}_j = \{i \mid M_{ij} = 1\} \text{ (non-zero rows)}$$

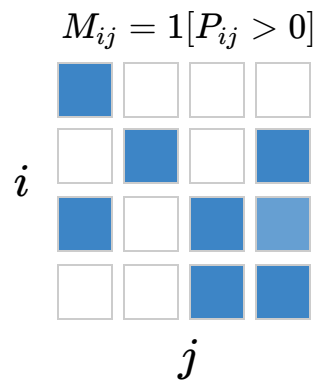
⚡ Efficient backward pass!

$$dQ_i = \sum_{j \in \mathcal{Q}_i} dS_i^{(j)} \cdot K_j$$

$$dK_j = \sum_{i \in \mathcal{K}_j} dS_i^{(j)} \cdot Q_i$$

$$dV_j = \sum_{i \in \mathcal{K}_j} P_i^{(j)} \cdot dO_i$$

⚡ AdaSplash: Adapative Sparse Flash Attention



$$\mathcal{Q}_i = \{j \mid M_{ij} = 1\} \text{ (non-zero cols)}$$

$$\mathcal{K}_j = \{i \mid M_{ij} = 1\} \text{ (non-zero rows)}$$

⚡ Efficient backward pass!

$$dQ_i = \sum_{j \in \mathcal{Q}_i} dS_i^{(j)} \cdot K_j$$

$$dK_j = \sum_{i \in \mathcal{K}_j} dS_i^{(j)} \cdot Q_i$$

$$dV_j = \sum_{i \in \mathcal{K}_j} P_i^{(j)} \cdot dO_i$$

$$\mathbf{J}_{\text{sparsemax}}(\mathbf{z}) \cdot \mathbf{v} = \mathbf{s} \odot (\mathbf{v} - \hat{v}\mathbf{1}), \text{ with } \hat{v} := \frac{\sum_{j \in S(\mathbf{z})} v_j}{|S(\mathbf{z})|}. \quad (14)$$

Interestingly, if $\text{sparsemax}(\mathbf{z})$ has already been evaluated (*i.e.*, in the forward step), then so has $S(\mathbf{z})$, hence the nonzeros of $\mathbf{J}_{\text{sparsemax}}(\mathbf{z}) \cdot \mathbf{v}$ can be computed in only $O(|S(\mathbf{z})|)$ time, which can be sublinear. This can be an important advantage of sparsemax over softmax if K is large.

(Martins and Astudillo, 2016)

⚡ AdaSplash:

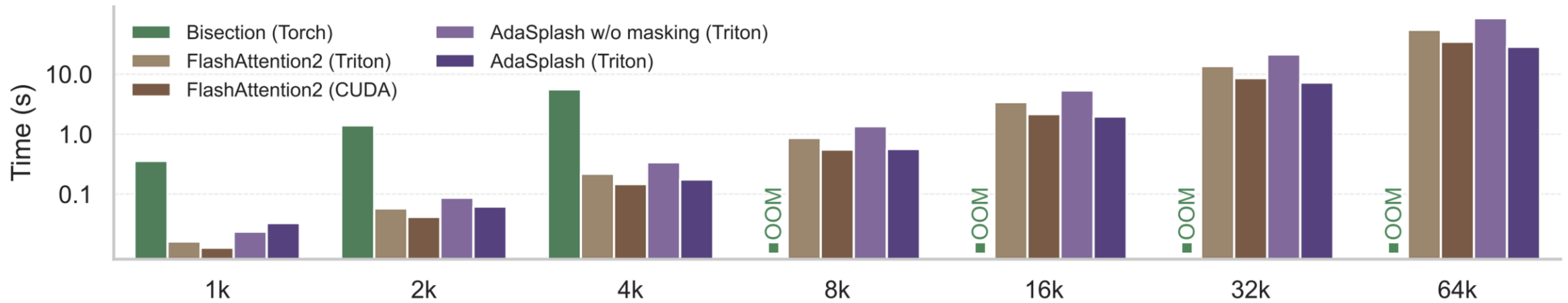


Figure 3. Efficiency of algorithms for computing non-causal attention in terms of the average training step time for increasingly longer sequence lengths. We use $\alpha = 1.5$ for α -entmax based methods (Bisection and ADASPLASH).

⚡ AdaSplash:

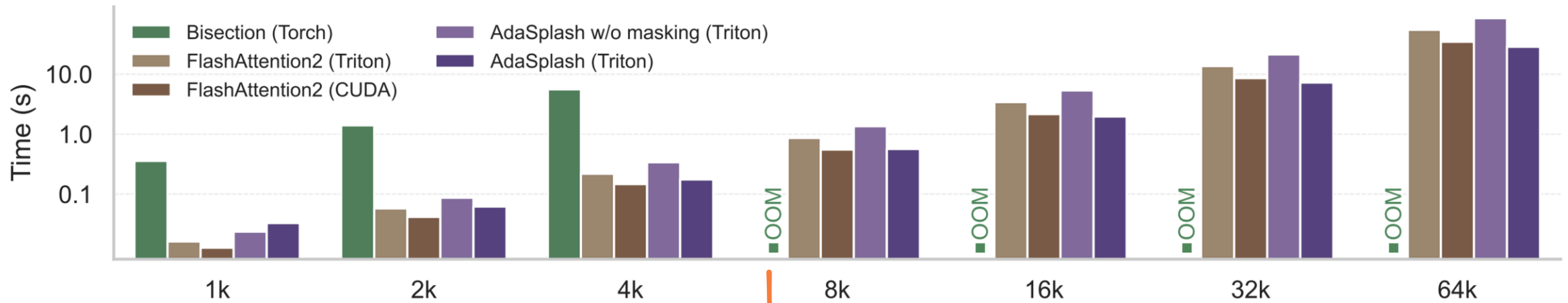


Figure 3. Efficiency of algorithms for computing non-causal attention in terms of the average training step time for increasingly longer sequence lengths. We use $\alpha = 1.5$ for α -entmax based methods (Bisection and ADASPLASH).

Before AdaSplash, entmax attention was not scalable!

⚡ AdaSplash:

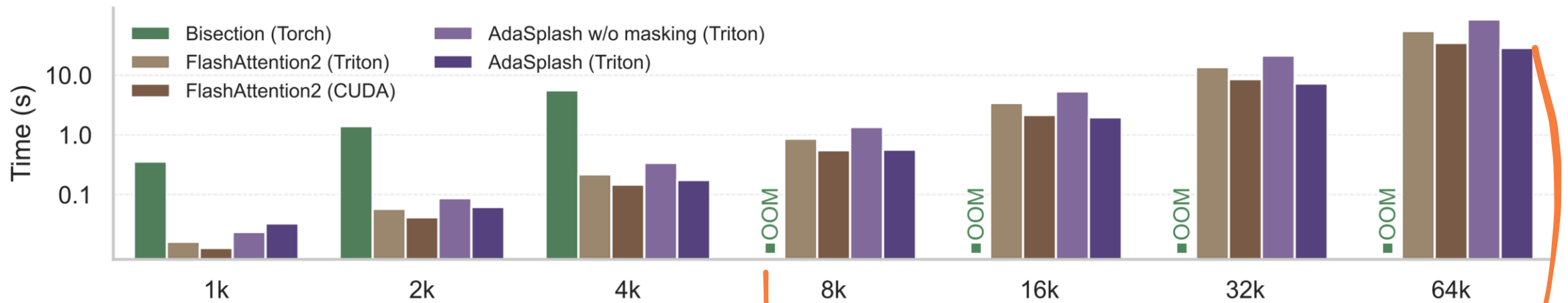


Figure 3. Efficiency of algorithms for computing non-causal attention in terms of the average training step time for increasingly longer sequence lengths. We use $\alpha = 1.5$ for α -entmax based methods (Bisection and ADASPLASH).

Before AdaSplash, entmax attention was not scalable!

A bit faster than FlashAttention!

⚡ ⚡ AdaSplash-2: Faster Differentiable Sparse Attention

w/



Nuno Gonçalves



Hugo Pitorro



Vlad Niculae



Edoardo Ponti



Lei Li



André F. T. Martins

⚡ ⚡ AdaSplash-2: Faster Differentiable Sparse Attention

w/



Nuno Gonçalves



Hugo Pitorro



Vlad Niculae



Edoardo Ponti



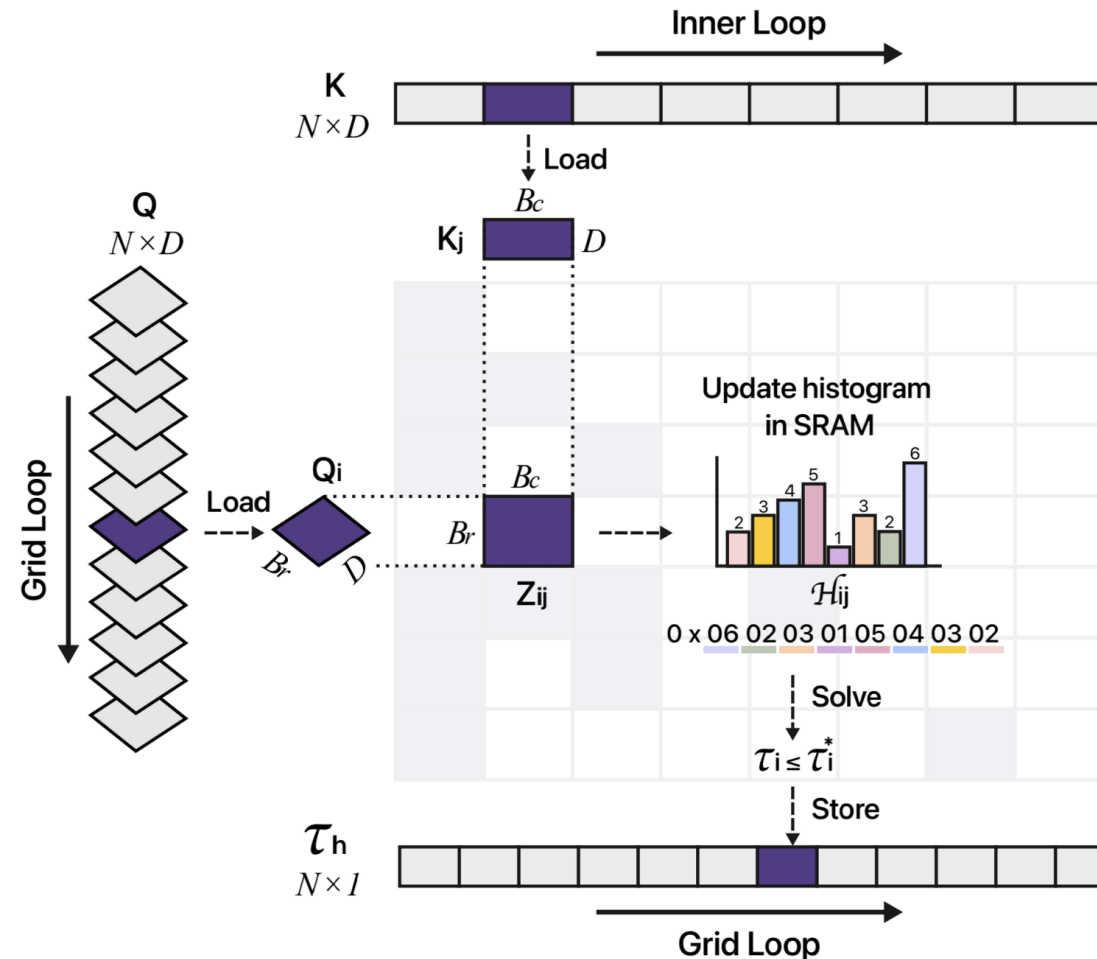
Lei Li



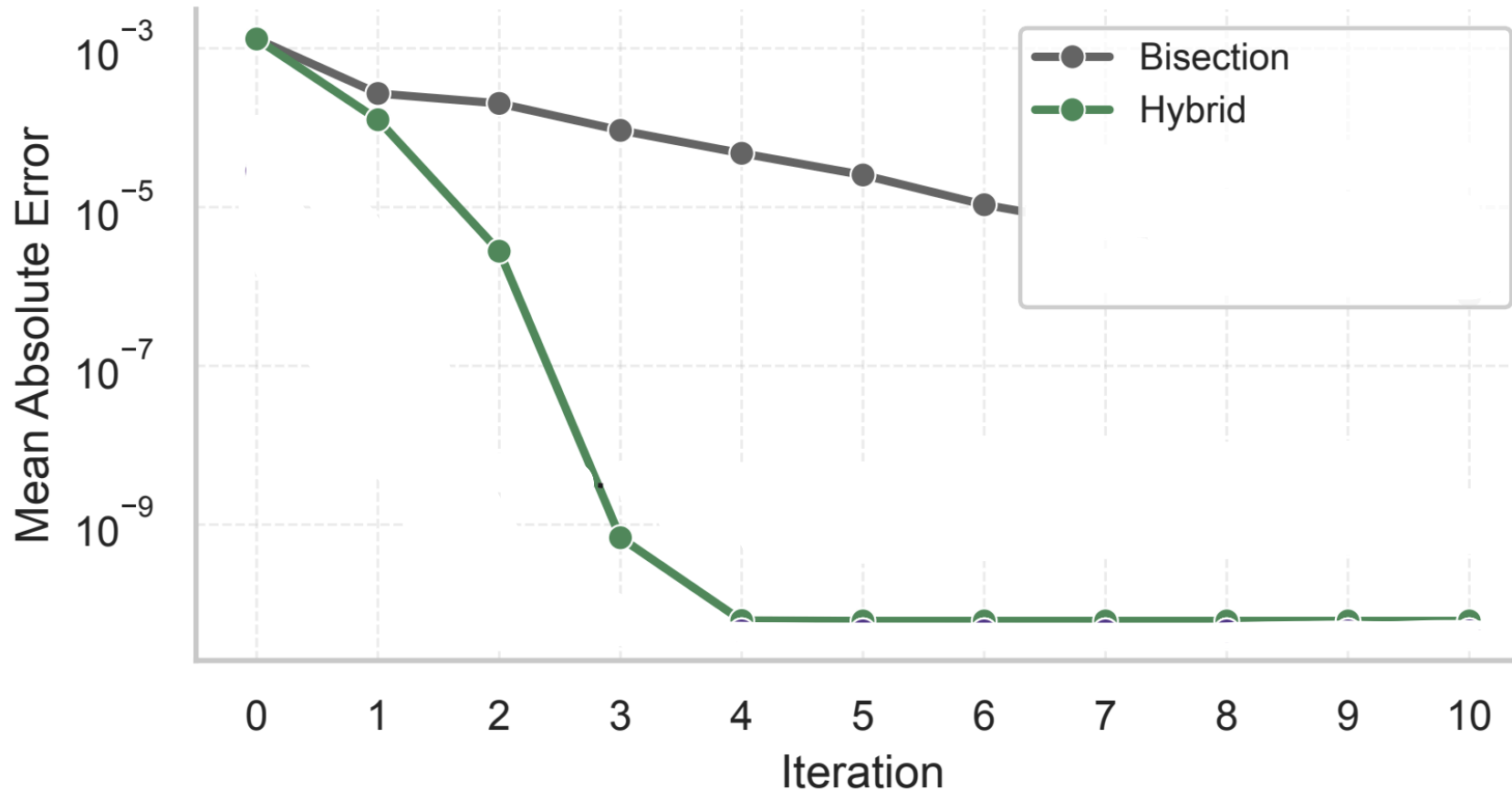
André F. T. Martins

- Halley-Bisection -> Halley-Newton-Secant-Bisection
- Histogram initialization for τ (lower bound)

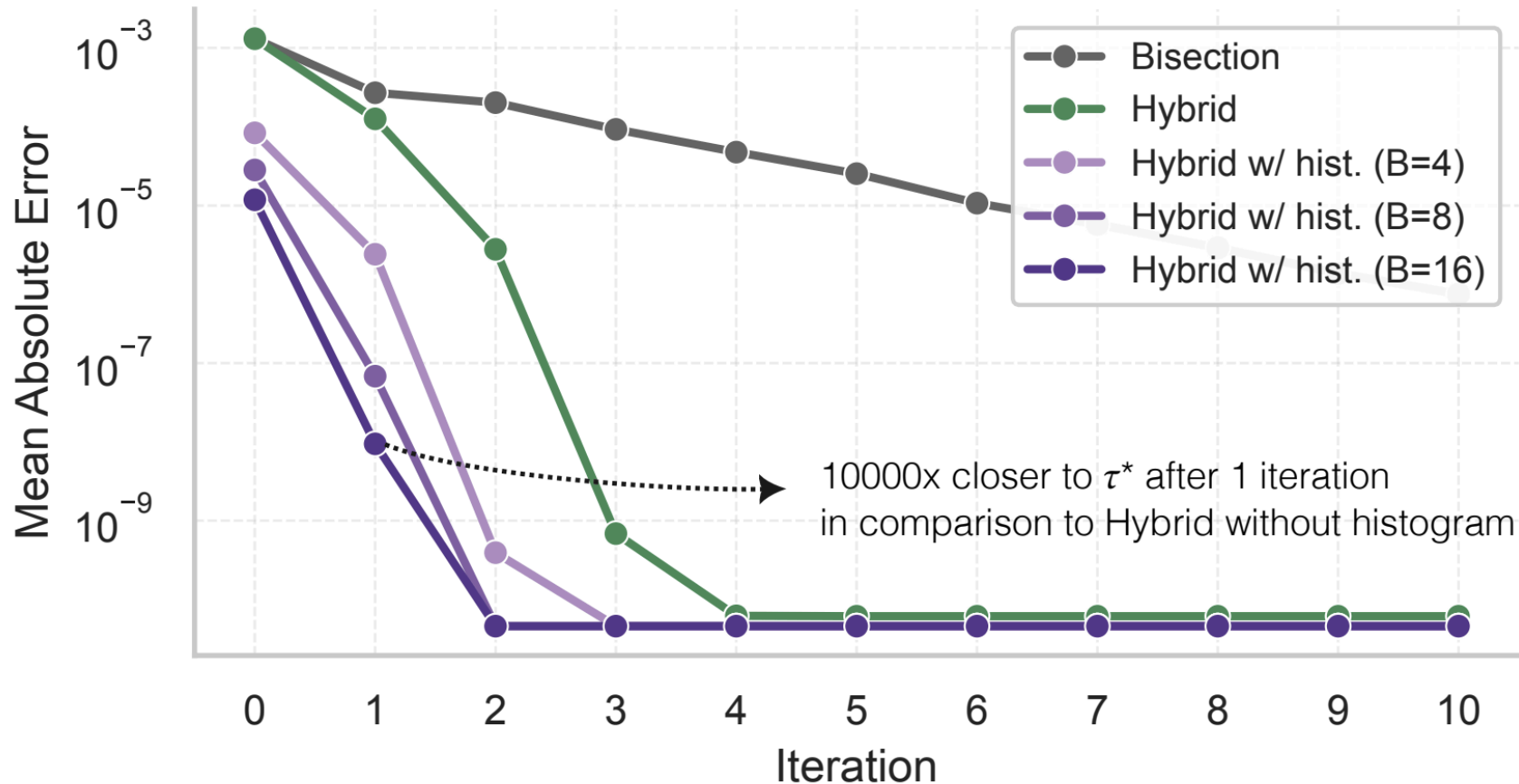
⚡ ⚡ AdaSplash-2: Faster Differentiable Sparse Attention



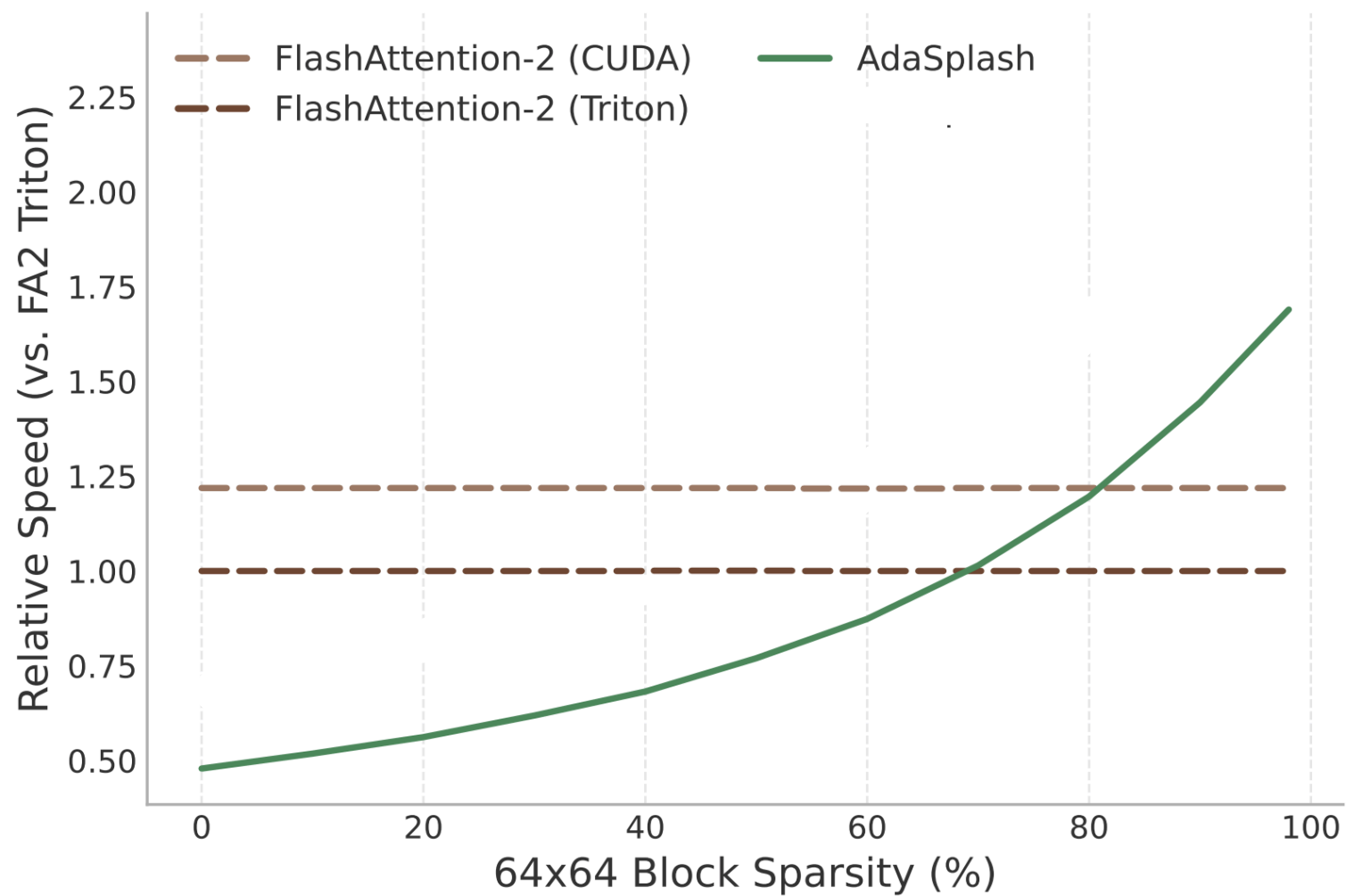
⚡ ⚡ AdaSplash-2: Faster Differentiable Sparse Attention



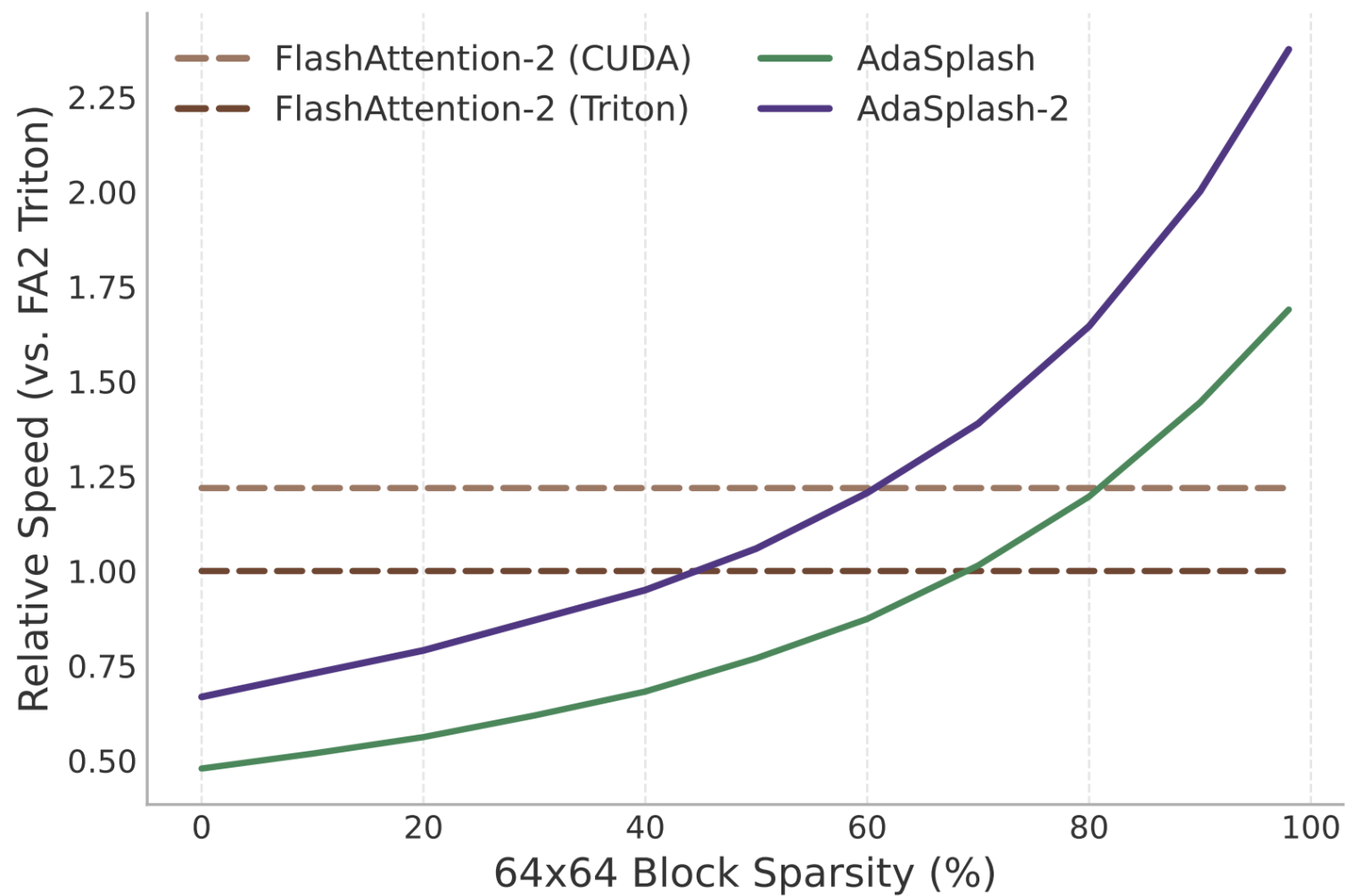
⚡ ⚡ AdaSplash-2: Faster Differentiable Sparse Attention



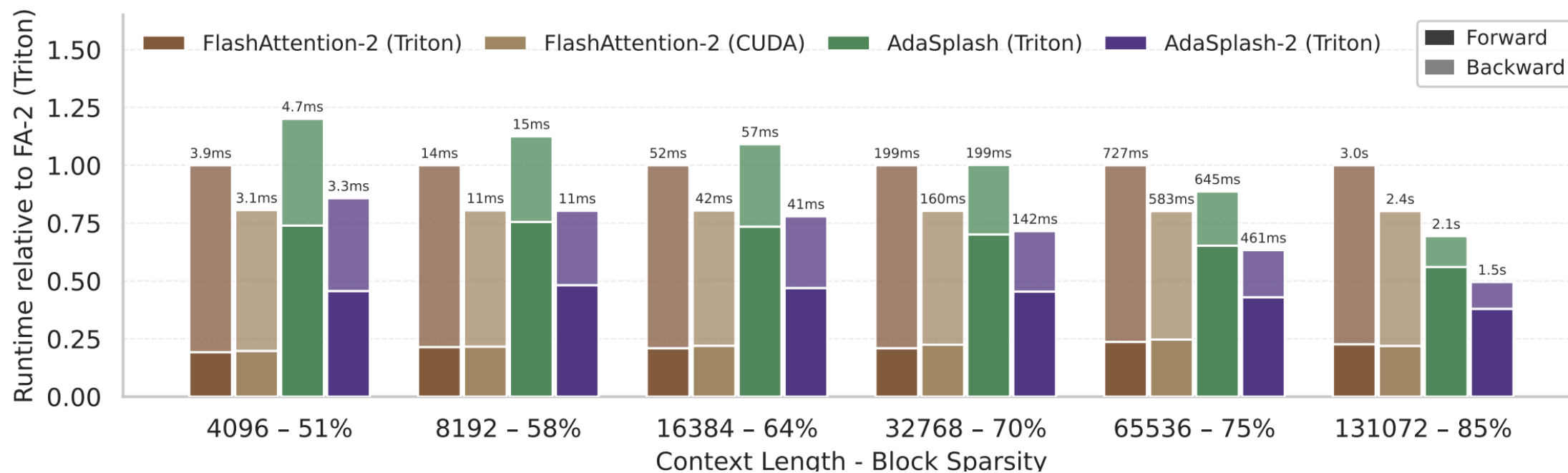
⚡ AdaSplash-1:



⚡ AdaSplash-2:



⚡ AdaSplash-2:



RULER - 32K context length

Table 1. RULER benchmark results for 1B parameter models trained up to 32K context-length. Best average results are in **bold**.

Model	Len.	MK-1	MK-2	MK-3	MQ	MV	NIAH-1	NIAH-2	NIAH-3	CWE	FWE	VT	QA-H	QA-S	Avg.
Softmax (RoPE)	4K	80.0	62.2	38.2	47.6	43.4	100.0	100.0	98.8	5.1	24.9	8.0	24.8	35.0	51.4
	8K	78.4	45.0	30.2	36.7	31.0	100.0	99.2	98.2	0.1	24.3	2.8	22.4	20.2	45.3
	16K	67.6	10.0	14.8	22.1	22.7	100.0	99.2	99.2	0.0	19.0	0.2	17.0	18.7	37.7
	32K	41.4	8.2	0.8	17.4	20.8	96.2	61.6	75.6	0.0	6.3	0.0	16.8	10.1	27.3
Entmax (RoPE)	4K	75.4	26.0	23.6	29.9	52.3	100.0	100.0	98.8	36.3	32.3	3.9	25.8	36.7	49.3
	8K	71.8	19.4	8.6	22.5	42.5	100.0	99.8	94.6	8.9	23.7	3.6	23.2	19.6	41.4
	16K	53.2	8.6	1.0	12.9	22.0	100.0	81.4	79.4	0.1	18.6	1.2	18.2	20.8	32.1
	32K	33.2	4.4	0.6	7.5	12.9	97.0	40.8	64.2	0.0	15.0	0.2	18.6	9.2	23.4
Softmax (NAPE)	4K	88.2	29.8	13.0	28.9	27.9	100.0	100.0	100.0	35.3	27.4	14.8	21.8	37.9	48.1
	8K	78.4	38.6	12.6	33.2	34.1	100.0	100.0	99.0	27.0	26.3	12.3	19.2	22.9	46.4
	16K	85.6	19.6	7.4	26.8	30.6	100.0	100.0	98.8	18.0	25.6	9.8	20.0	21.3	43.3
	32K	66.6	4.2	2.0	26.4	26.5	100.0	97.0	98.0	8.3	3.3	7.7	19.0	19.5	36.8
Entmax (NAPE)	4K	81.2	12.2	27.2	51.9	55.6	100.0	100.0	79.8	48.9	58.1	31.4	30.2	34.5	54.7
	8K	73.4	4.6	17.6	62.3	31.5	100.0	100.0	84.0	33.7	48.3	38.4	29.0	26.0	49.9
	16K	79.2	1.4	9.8	38.7	24.0	100.0	100.0	74.2	22.9	51.1	34.5	27.2	27.5	45.4
	32K	63.0	1.8	4.8	25.8	16.8	100.0	92.0	76.0	11.7	45.8	27.4	25.0	22.1	39.4

RULER - 32K context length

Table 1. RULER benchmark results for 1B parameter models trained up to 32K context-length. Best average results are in **bold**.

Model	Len.	MK-1	MK-2	MK-3	MQ	MV	NIAH-1	NIAH-2	NIAH-3	CWE	FWE	VT	QA-H	QA-S	Avg.
Softmax (RoPE)	4K	80.0	62.2	38.2	47.6	43.4	100.0	100.0	98.8	5.1	24.9	8.0	24.8	35.0	51.4
	8K	78.4	45.0	30.2	36.7	31.0	100.0	99.2	98.2	0.1	24.3	2.8	22.4	20.2	45.3
	16K	67.6	10.0	14.8	22.1	22.7	100.0	99.2	99.2	0.0	19.0	0.2	17.0	18.7	37.7
	32K	41.4	8.2	0.8	17.4	20.8	96.2	61.6	75.6	0.0	6.3	0.0	16.8	10.1	27.3
Entmax (RoPE)	4K	75.4	26.0	23.6	29.9	52.3	100.0	100.0	98.8	36.3	32.3	3.9	25.8	36.7	49.3
	8K	71.8	19.4	8.6	22.5	42.5	100.0	99.8	94.6	8.9	23.7	3.6	23.2	19.6	41.4
	16K	53.2	8.6	1.0	12.9	22.0	100.0	81.4	79.4	0.1	18.6	1.2	18.2	20.8	32.1
	32K	33.2	4.4	0.6	7.5	12.9	97.0	40.8	64.2	0.0	15.0	0.2	18.6	9.2	23.4
Softmax (NAPE)	4K	88.2	29.8	13.0	28.9	27.9	100.0	100.0	100.0	35.3	27.4	14.8	21.8	37.9	48.1
	8K	78.4	38.6	12.6	33.2	34.1	100.0	100.0	99.0	27.0	26.3	12.3	19.2	22.9	46.4
	16K	85.6	19.6	7.4	26.8	30.6	100.0	100.0	98.8	18.0	25.6	9.8	20.0	21.3	43.3
	32K	66.6	4.2	2.0	26.4	26.5	100.0	97.0	98.0	8.3	3.3	7.7	19.0	19.5	36.8
Entmax (NAPE)	4K	81.2	12.2	27.2	51.9	55.6	100.0	100.0	79.8	48.9	58.1	31.4	30.2	34.5	54.7
	8K	73.4	4.6	17.6	62.3	31.5	100.0	100.0	84.0	33.7	48.3	38.4	29.0	26.0	49.9
	16K	79.2	1.4	9.8	38.7	24.0	100.0	100.0	74.2	22.9	51.1	34.5	27.2	27.5	45.4
	32K	63.0	1.8	4.8	25.8	16.8	100.0	92.0	76.0	11.7	45.8	27.4	25.0	22.1	39.4

CWE = Common Words Extraction

FWE = Frequent Words Extraction

VT = Variable Tracking

**super relevant for coding
and agentic tasks!**

HELMET – In Context Learning

Table 2. Results across context lengths for the In-Context Learning tasks in the HELMET benchmark. Best average result is in **bold**.

Model	Len.	TREC-C	TREC-F	NLU	B77	C150	Avg.
Softmax (RoPE)	8K	62.4	33.6	12.6	6.0	13.6	25.6
	16K	67.2	42.2	26.4	10.8	25.8	34.5
	32K	69.2	47.8	25.8	12.8	35.6	38.2
Entmax (RoPE)	8K	63.2	26.6	25.4	8.4	20.2	28.8
	16K	66.4	35.8	32.8	13.8	27.4	35.2
	32K	66.8	42.8	36.2	16.8	34.0	39.3
Softmax (NAPE)	8K	58.8	30.8	27.2	13.4	32.6	32.6
	16K	64.4	35.8	38.6	20.6	43.2	40.5
	32K	71.2	45.6	41.4	21.2	51.0	46.1
Entmax (NAPE)	8K	78.0	43.8	44.6	19.4	41.0	45.4
	16K	83.4	56.4	58.0	25.0	57.0	56.0
	32K	85.0	62.4	60.6	34.4	68.4	62.2

HELMET – In Context Learning

Table 2. Results across context lengths for the In-Context Learning tasks in the HELMET benchmark. Best average result is in **bold**.

Model	Len.	TREC-C	TREC-F	NLU	B77	C150	Avg.
Softmax (RoPE)	8K	62.4	33.6	12.6	6.0	13.6	25.6
	16K	67.2	42.2	26.4	10.8	25.8	34.5
	32K	69.2	47.8	25.8	12.8	35.6	38.2
Entmax (RoPE)	8K	63.2	26.6	25.4	8.4	20.2	28.8
	16K	66.4	35.8	32.8	13.8	27.4	35.2
	32K	66.8	42.8	36.2	16.8	34.0	39.3
Softmax (NAPE)	8K	58.8	30.8	27.2	13.4	32.6	32.6
	16K	64.4	35.8	38.6	20.6	43.2	40.5
	32K	71.2	45.6	41.4	21.2	51.0	46.1
Entmax (NAPE)	8K	78.0	43.8	44.6	19.4	41.0	45.4
	16K	83.4	56.4	58.0	25.0	57.0	56.0
	32K	85.0	62.4	60.6	34.4	68.4	62.2

super relevant for RAG systems!

How sparse is the model in the end?

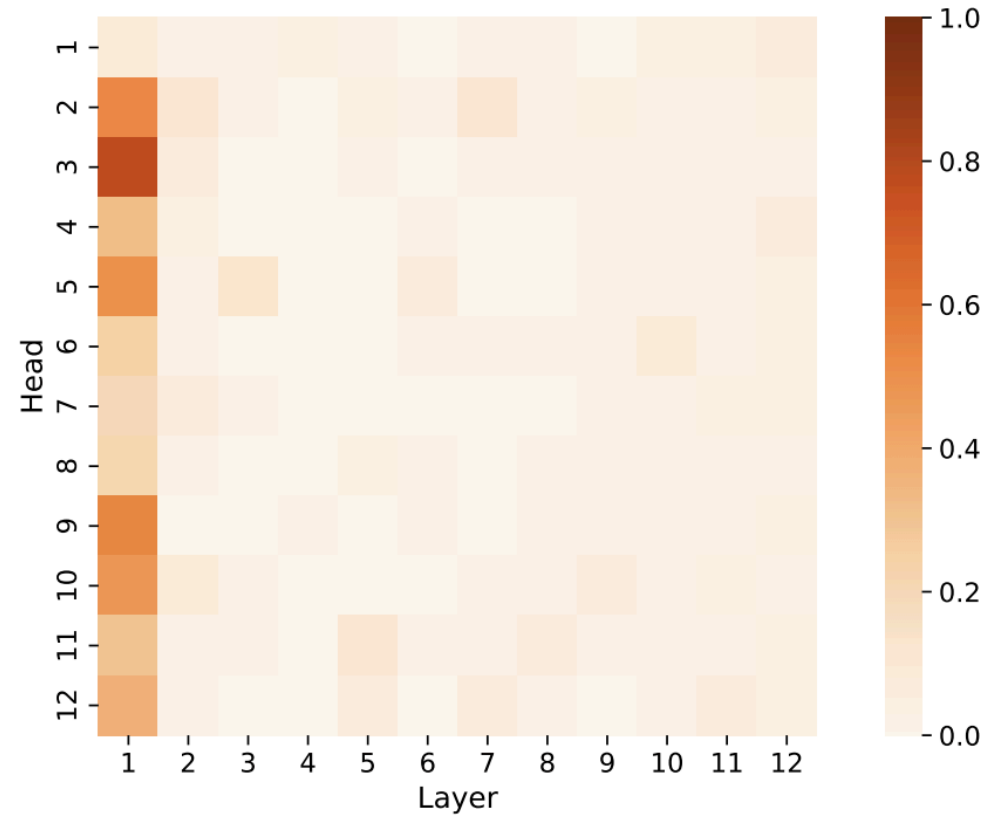


Figure 4. Ratio of non-zero attention scores for GPT-2 ($\alpha = 1.5$).

Long-context Generalization with Sparse Attention

w/



Pavlo Vasylenko



Hugo Pitorro

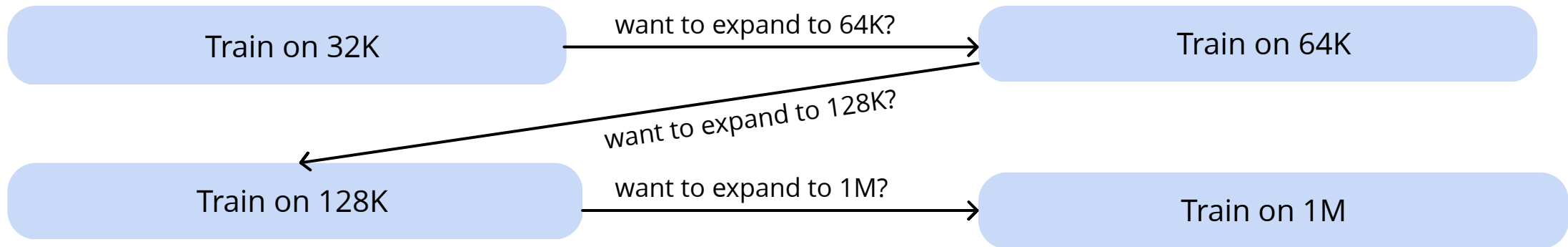


André F. T. Martins

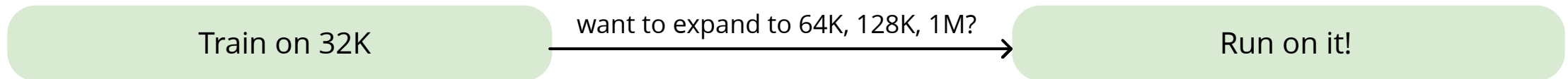
What happens with attention on sequences longer than those seen during training?

Long-context Generalization

path of pain 🙄



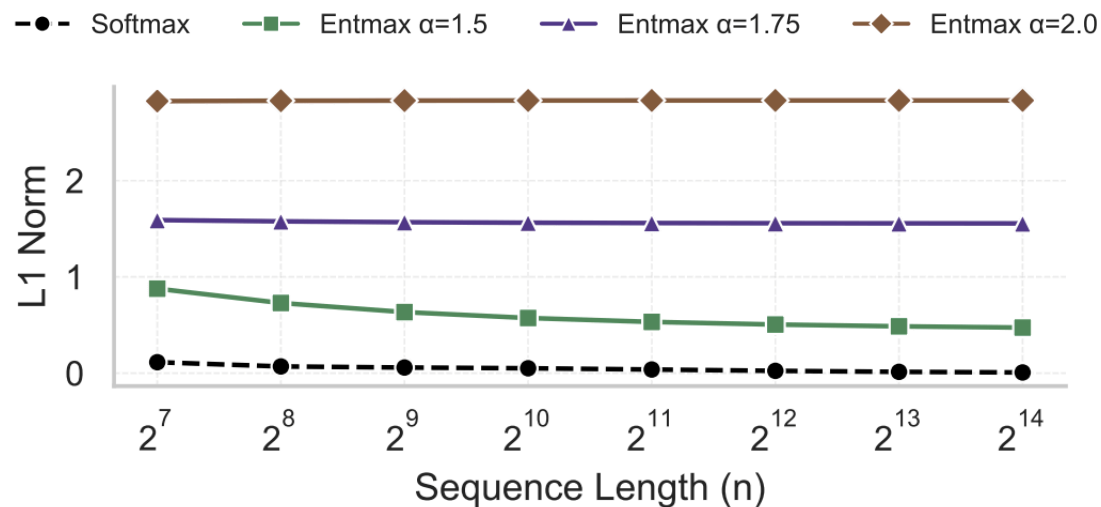
path of joy 😎



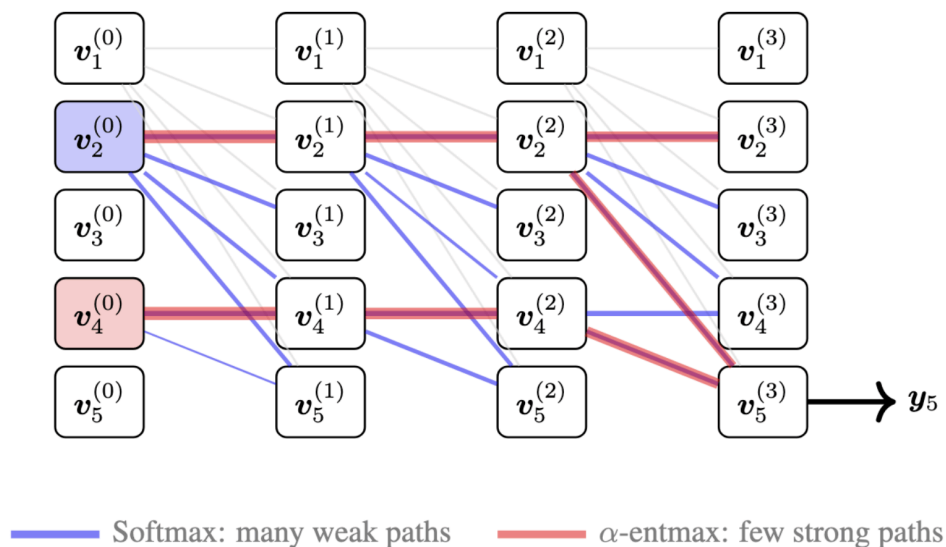
Theoretical Findings

(full statements and proofs in the paper)

✓ Preserve representations

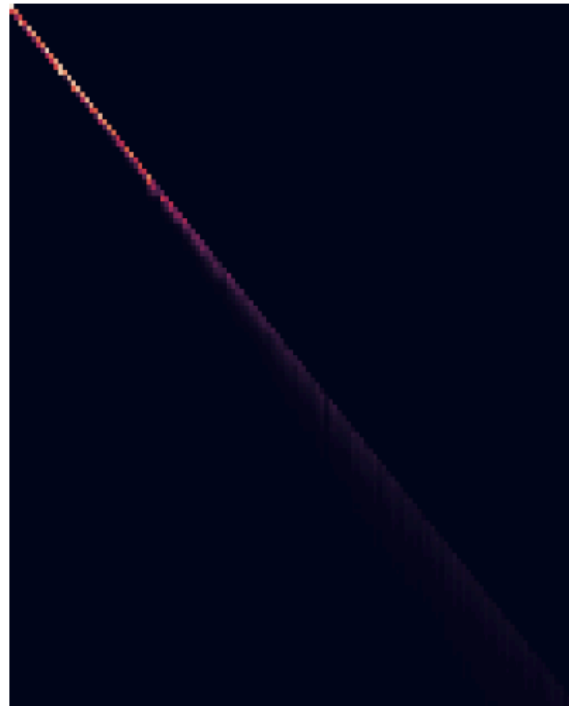


✓ Alleviate over-squashing

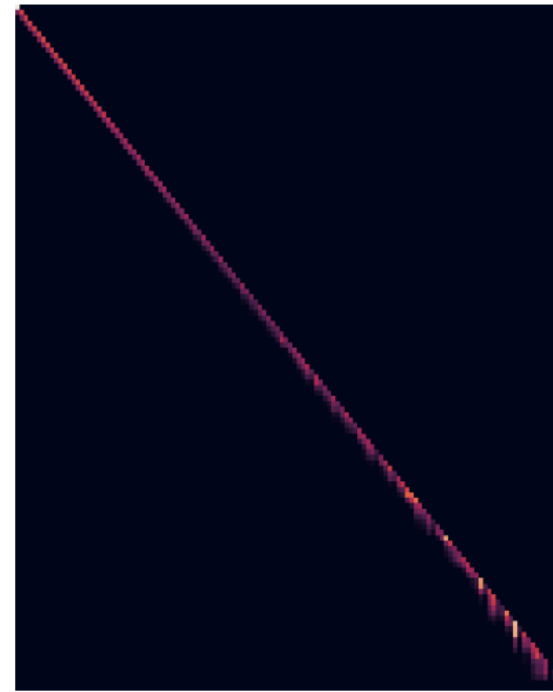


Theoretical Findings

✓ Maintain focus (non-dispersion)



Softmax



Entmax

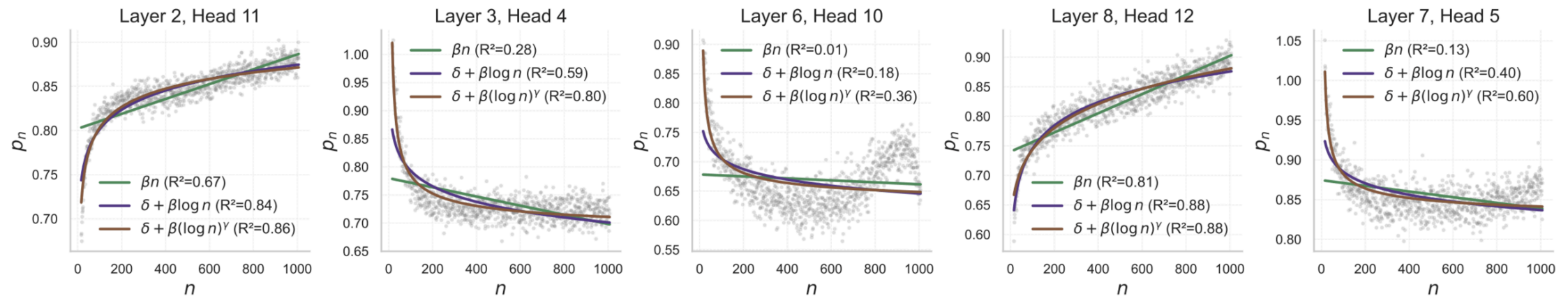
Adaptive-Scalable α -entmax (ASEntmax)

What if entmax ignores *too many* relevant tokens?

$$\text{ASEntmax}(\mathbf{Z}) = \alpha\text{-entmax}(\underbrace{(\delta + \beta(\log n)^\gamma)}_{\text{scaling to counteract excessive sparsification}})\mathbf{z}$$

scaling to counteract
excessive sparsification

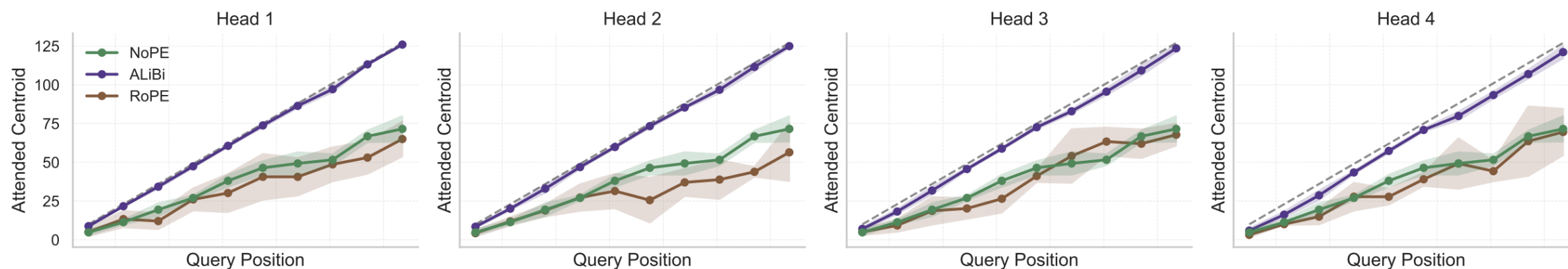
δ, β, γ are *learnable* for each *head* and *query*.



Positional Encoding for Sparse Attention

RoPE + α -entmax = hard, periodic, frequency-specific attention windows

⇒ hard cutoffs and "dead zones" that break long-range links

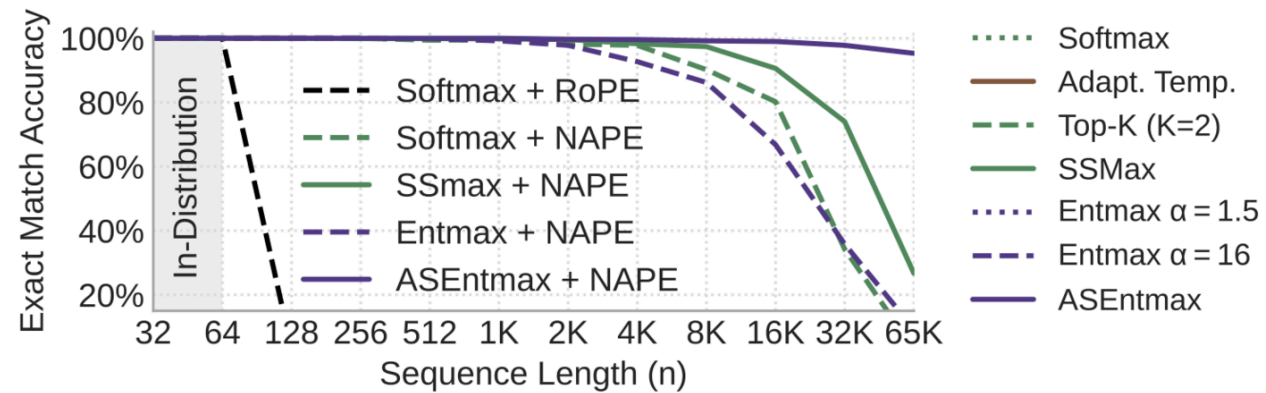


NAPE (NoPE + ALiBi) ⇒ half of the heads use NoPE, the other half use ALiBi

- short-span focus
- semantic-guided attention

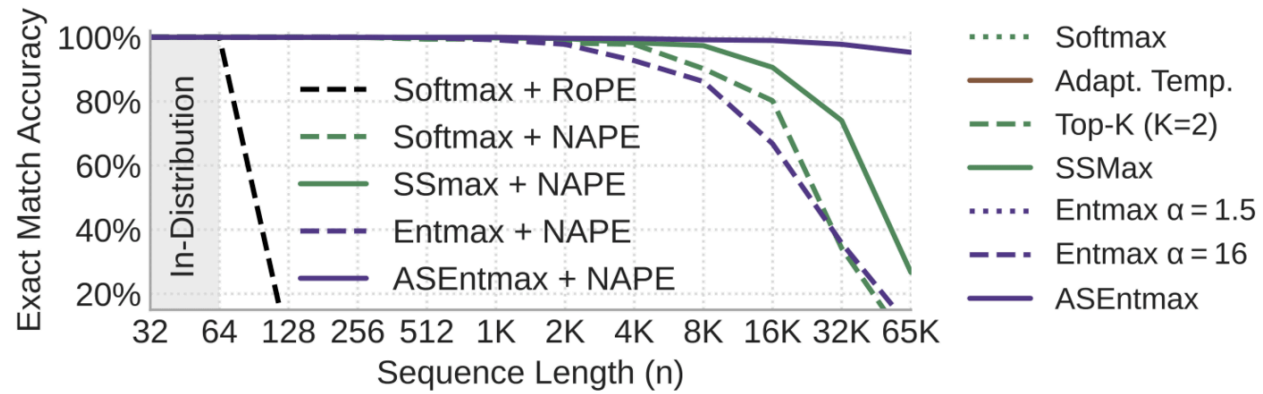
Results

Multi-query Multi-token Associative Recall



Results

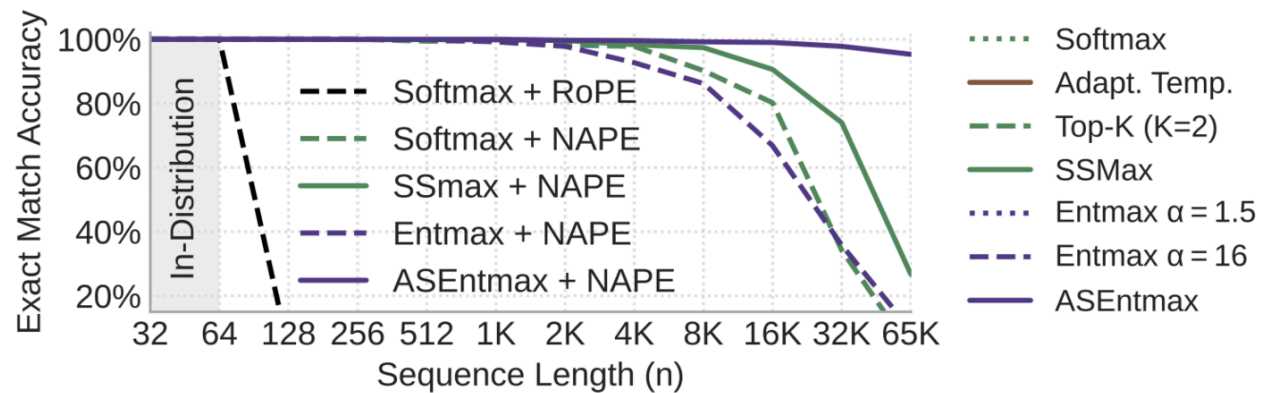
Multi-query Multi-token Associative Recall



1000x extrapolation!

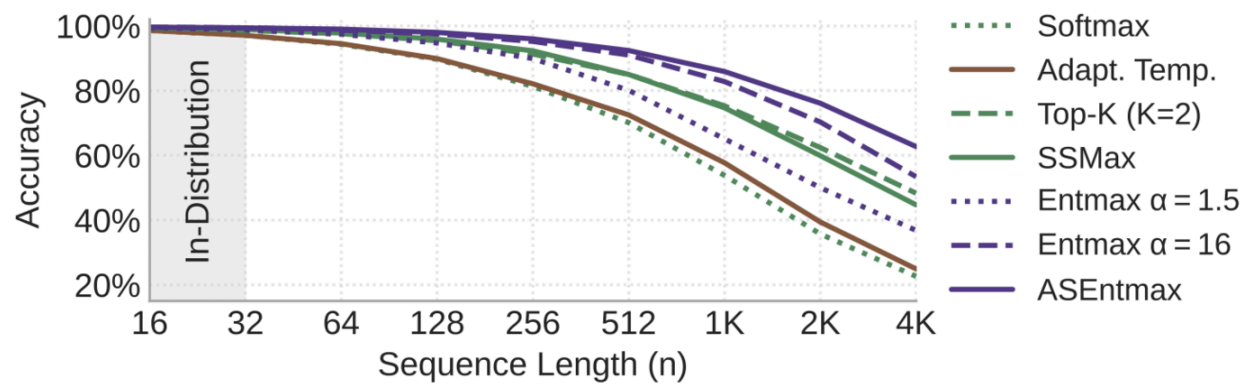
Results

Multi-query Multi-token Associative Recall



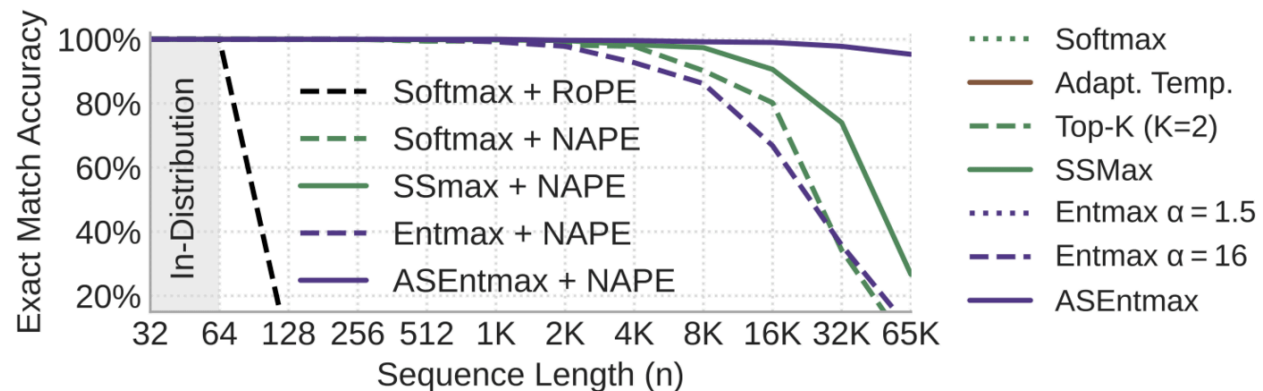
1000x extrapolation!

Max Retrieval



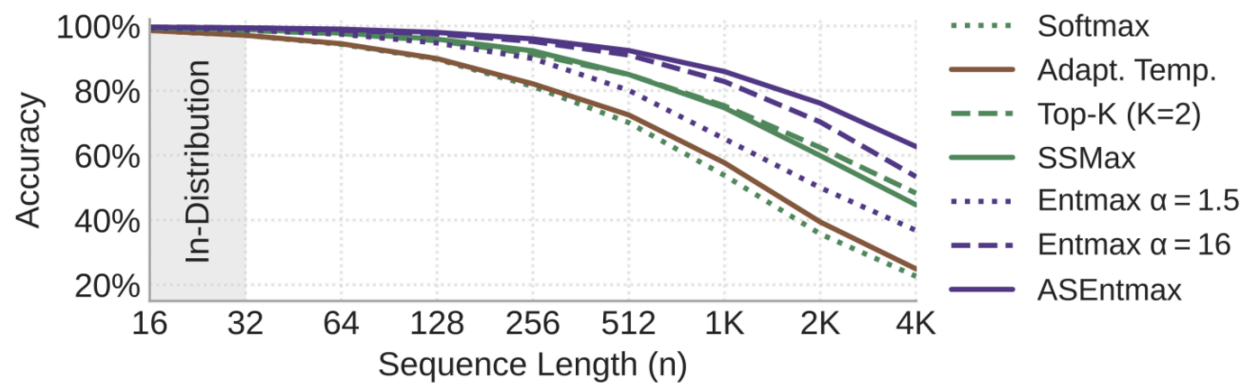
Results

Multi-query Multi-token Associative Recall



1000x extrapolation!

Max Retrieval



128x extrapolation!

Results

Length extrapolation on language modeling

Model	ArXiv (ID)		ArXiv (OOD)			PubMed (ID)		PubMed (OOD)		
	1K	2K	4K	8K	16K	1K	2K	4K	8K	16K
Softmax	20.82	16.47	13.74	12.69	12.70	18.36	17.34	15.31	15.61	18.73
SSMax	20.81	16.44	13.41	11.95	12.00	18.13	17.10	14.87	14.11	15.36
Entmax	23.40	18.57	15.36	14.02	14.68	22.33	21.14	18.94	18.31	20.29
ASEntmax	20.64	16.30	13.15	11.19	10.77	18.15	17.09	14.66	13.48	14.48

Results

RULER (needle in a haystack) with 350M models

Model	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	100.0	99.4	94.2	11.4	0.8	100.0	100.0	4.8	0.0
SSMax	100.0	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	100.0	100.0	99.8	97.4	99.4	99.4	83.2	25.4

Results

RULER (needle in a haystack) with 350M models

Model	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	100.0	99.4	94.2	11.4	0.8	100.0	100.0	4.8	0.0
SSMax	100.0	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	100.0	100.0	99.8	97.4	99.4	99.4	83.2	25.4

Comparing with the recent Mamba-3

Model (1.5B)	NIAH-Single-1			NIAH-Single-2			
	Context Length	1024	2048	4096	1024	2048	4096
Transformer		100.0	100.0	0.0	92.2	100.0	0.0
Gated DeltaNet		100.0	100.0	99.8	100.0	<u>93.8</u>	<u>49.8</u>
Mamba-2		100.0	99.6	62.0	100.0	<u>53.8</u>	<u>11.8</u>
Mamba-3		100.0	100.0	<u>88.2</u>	100.0	95.4	50.6

Results

RULER (needle in a haystack) with 350M models

Model	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	100.0	99.4	94.2	11.4	0.8	100.0	100.0	4.8	0.0
SSMax	100.0	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	100.0	100.0	99.8	97.4	99.4	99.4	83.2	25.4

Comparing with the recent Mamba-3

Model (1.5B)	NIAH-Single-1			NIAH-Single-2			
	Context Length	1024	2048	4096	1024	2048	4096
Transformer		100.0	100.0	0.0	92.2	100.0	0.0
Gated DeltaNet		100.0	100.0	99.8	100.0	93.8	49.8
Mamba-2		100.0	99.6	62.0	100.0	53.8	11.8
Mamba-3		100.0	100.0	88.2	100.0	95.4	50.6

Results

RULER (needle in a haystack) with 350M models

Model	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	100.0	99.4	94.2	11.4	0.8	100.0	100.0	4.8	0.0
SSMax	100.0	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	100.0	100.0	99.8	97.4	99.4	99.4	83.2	25.4

Comparing with the recent Mamba-3

Model (1.5B)	NIAH-Single-1			NIAH-Single-2			
	Context Length	1024	2048	4096	1024	2048	4096
Transformer		100.0	100.0	0.0	92.2	100.0	0.0
Gated DeltaNet		100.0	100.0	99.8	100.0	93.8	49.8
Mamba-2		100.0	99.6	62.0	100.0	53.8	11.8
Mamba-3		100.0	100.0	88.2	100.0	95.4	50.6

AdaSplash for inference?

- Inference with transformers is a different systems problem
 - Prefill (compute bound) vs Generation (memory bound)
- Two ongoing directions!

1. Use (good) approximations at decoding

EntmaxKV: Support-Aware Decoding for Entmax Attention

Gonçalo Duarte¹, Miguel Couceiro^{1,2,3}, Marcos V. Treviso^{1,2,4}

¹Instituto Superior Técnico, Universidade de Lisboa. ²ELLIS Unit Lisbon.

³INESC-ID. ⁴Instituto de Telecomunicações.

2. Decompose attention in a hierarchy

DashAttention: Differentiable and Adaptive Sparse Hierarchical Attention

**Yuxiang Huang^{1*}, Nuno M. T. Gonçalves^{2,3,4*}, Federico Alvetreti⁵, Lei Li⁴,
Xu Han¹, Edoardo M. Ponti⁶, André F. T. Martins^{2,3,7,8}, Marcos V. Treviso^{2,3,7}**

¹Tsinghua University. ²Instituto Superior Técnico, Universidade de Lisboa.

³Instituto de Telecomunicações. ⁴Carnegie Mellon University. ⁵Sapienza University of Rome.

⁶University of Edinburgh. ⁷TransPerfect. ⁸ELLIS Unit Lisbon.

huang-yx21@mails.tsinghua.edu.cn, nuno.m.goncalves@ulisboa.pt

EntmaxKV

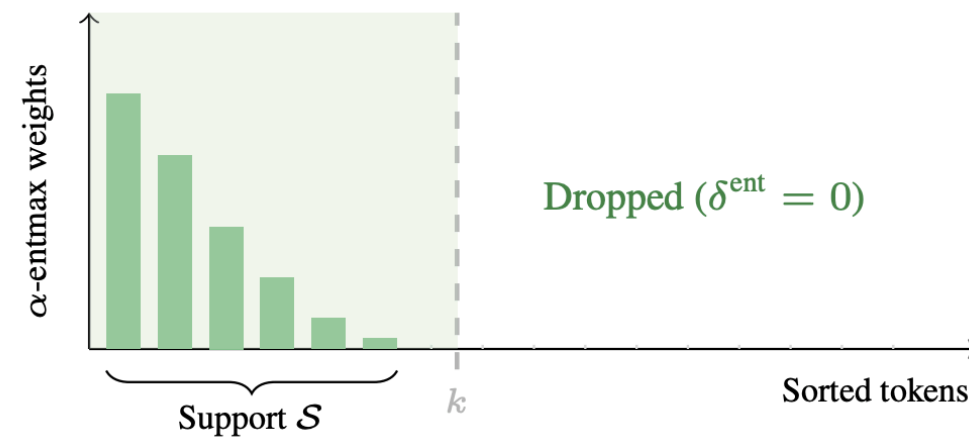
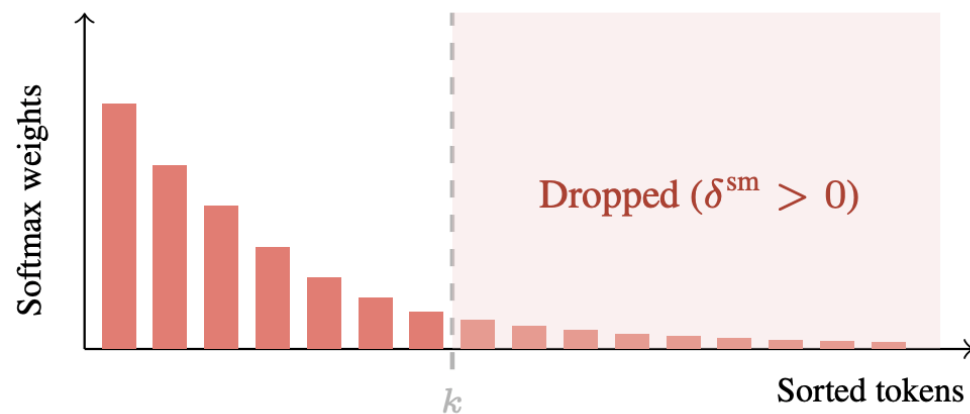
w/



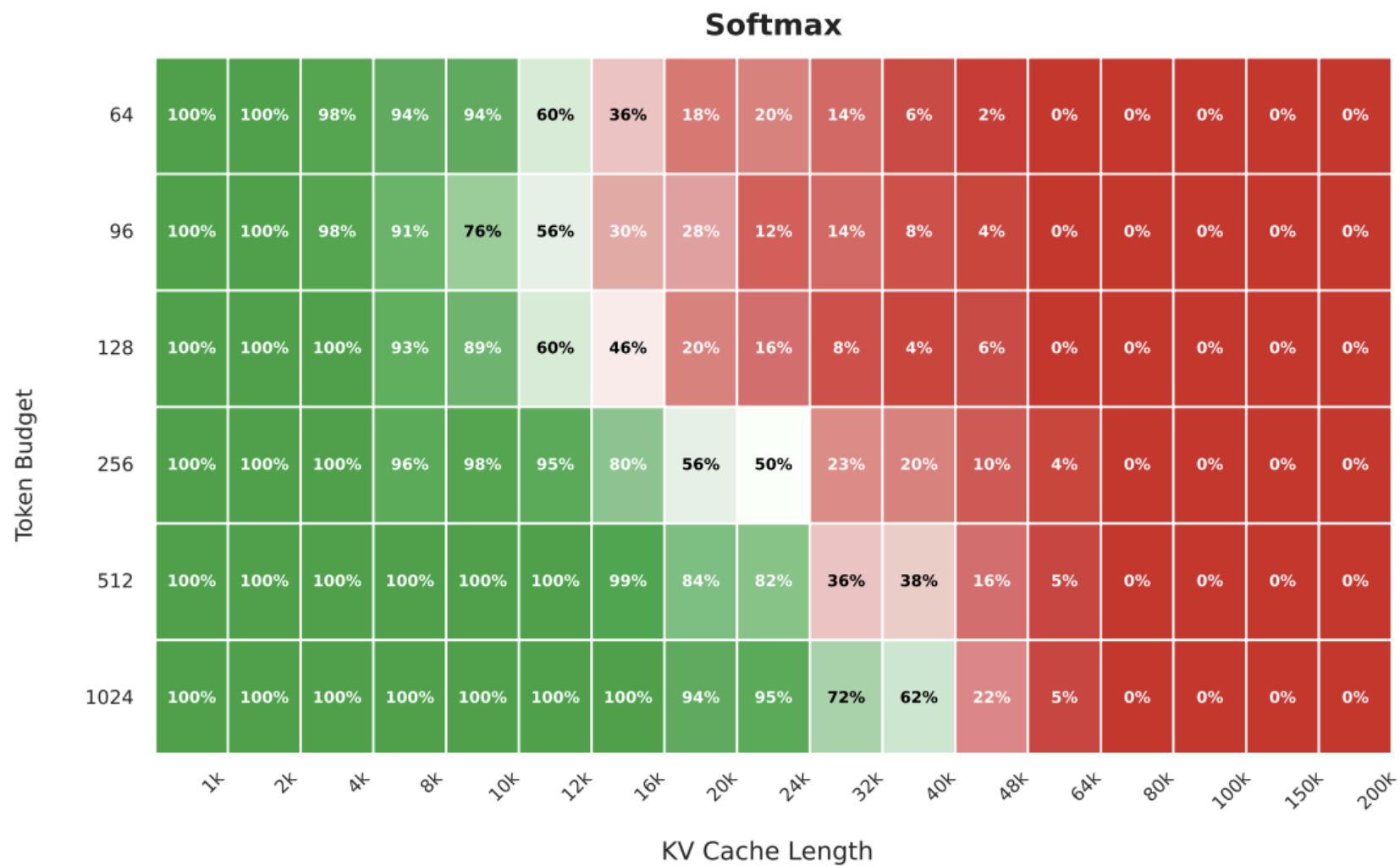
Gonçalo Duarte



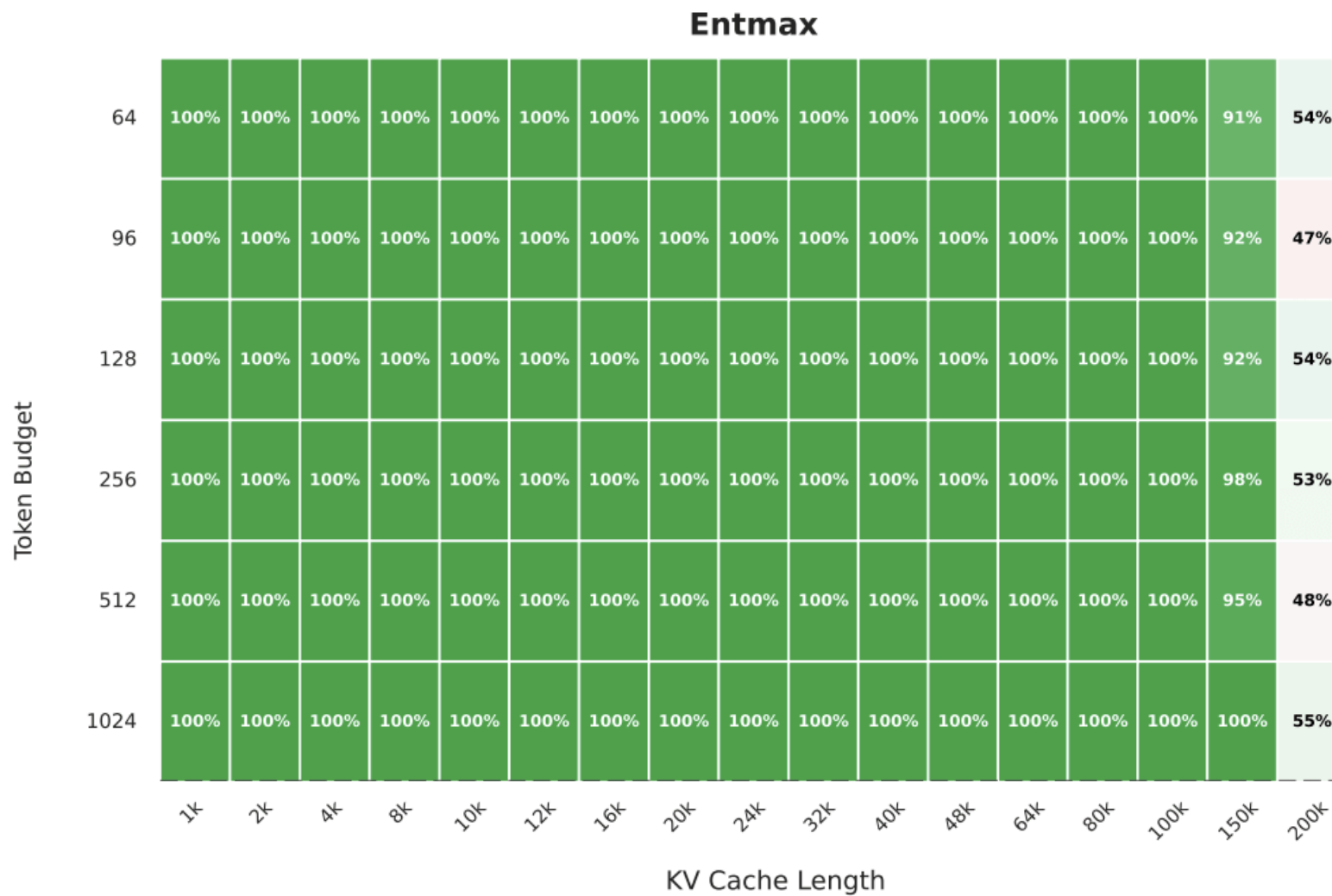
Miguel Couceiro



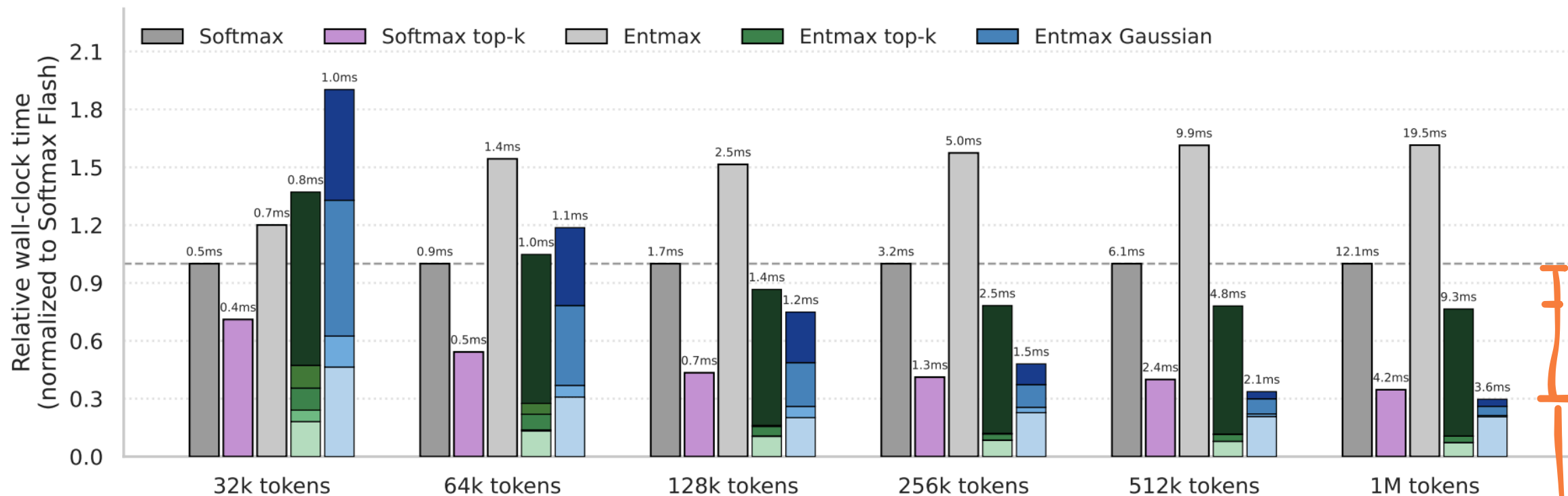
EntmaxKV



EntmaxKV



EntmaxKV



Gaussian-approximated version is **~3x** faster than FlashDecoding

Top-k version is **~1.25x** faster than FlashDecoding

Dash Attention

w/



Yuxiang Haung



Nuno Gonçalves



Federico Alvetreti



Lei Li



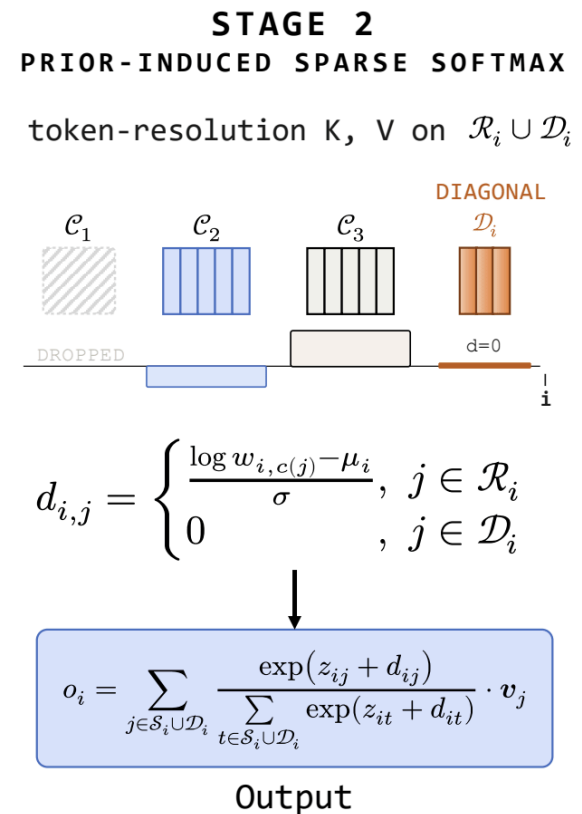
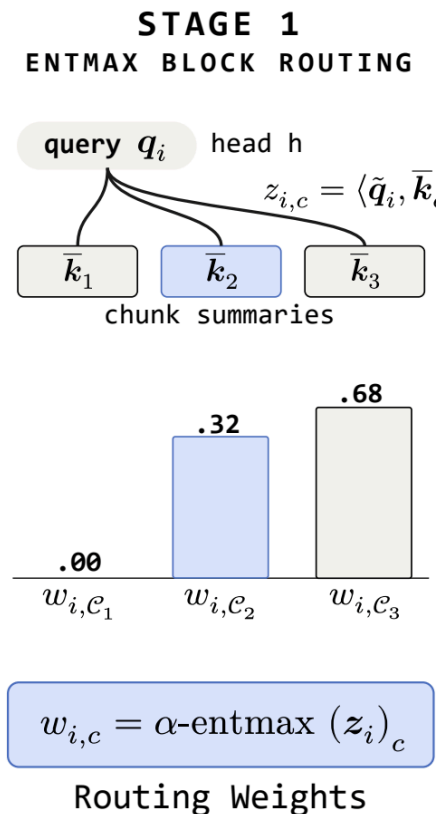
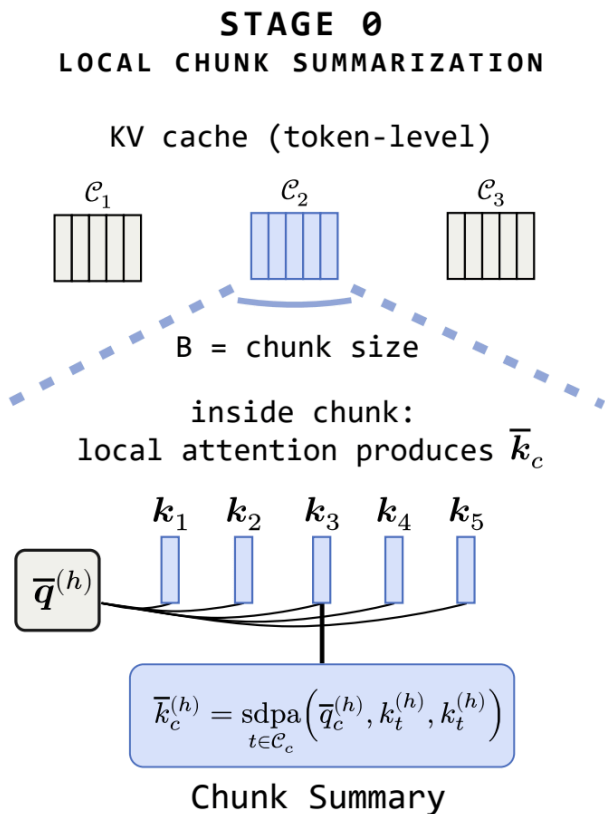
Xu Han



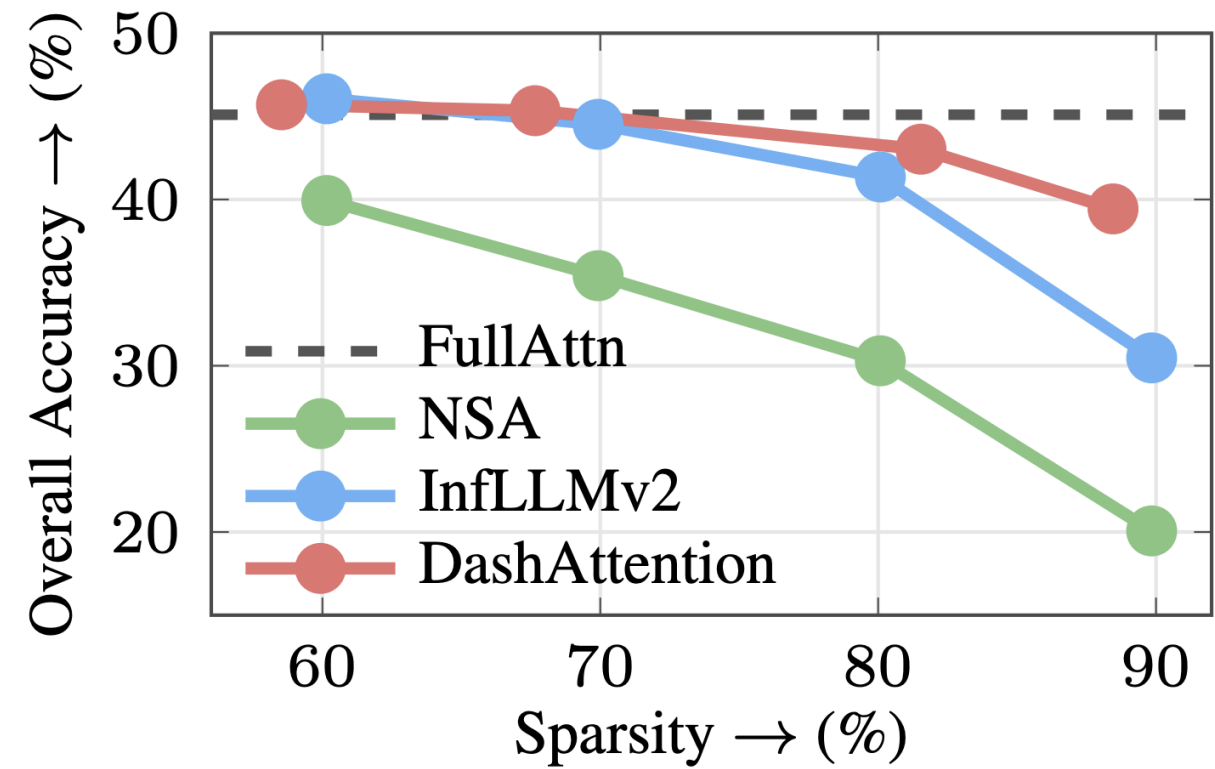
Edoardo Ponti



André Martins



Dash Attention



Dash Attention

<i>N</i>	PREFILL									DECODING								
	<i>s</i> = 75%			<i>s</i> = 87.5%			<i>s</i> = 93.7%			<i>s</i> = 75%			<i>s</i> = 87.5%			<i>s</i> = 93.7%		
	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA
16K	0.71	0.96	1.34	1.23	1.63	2.09	1.91	2.40	2.87	0.40	1.20	1.54	0.43	1.43	1.67	0.46	1.73	1.93
32K	0.76	1.04	1.39	1.25	1.67	2.11	2.19	2.78	2.94	0.57	1.46	1.69	0.65	1.85	2.26	0.73	2.29	2.69
64K	0.82	1.01	1.55	1.33	1.77	2.28	2.24	2.86	3.02	0.71	1.59	1.97	0.95	2.28	2.69	1.12	2.92	3.27
96K	0.82	1.04	1.63	1.31	1.73	2.34	2.32	3.06	3.09	0.80	1.73	1.96	1.04	2.38	2.72	1.34	3.10	3.36

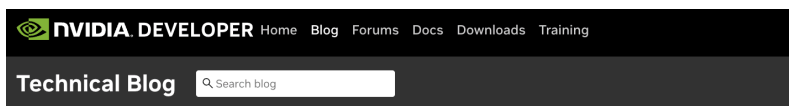
Dash Attention

<i>N</i>	PREFILL									DECODING								
	<i>s</i> = 75%			<i>s</i> = 87.5%			<i>s</i> = 93.7%			<i>s</i> = 75%			<i>s</i> = 87.5%			<i>s</i> = 93.7%		
	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA
16K	0.71	0.96	1.34	1.23	1.63	2.09	1.91	2.40	2.87	0.40	1.20	1.54	0.43	1.43	1.67	0.46	1.73	1.93
32K	0.76	1.04	1.39	1.25	1.67	2.11	2.19	2.78	2.94	0.57	1.46	1.69	0.65	1.85	2.26	0.73	2.29	2.69
64K	0.82	1.01	1.55	1.33	1.77	2.28	2.24	2.86	3.02	0.71	1.59	1.97	0.95	2.28	2.69	1.12	2.92	3.27
96K	0.82	1.04	1.63	1.31	1.73	2.34	2.32	3.06	3.09	0.80	1.73	1.96	1.04	2.38	2.72	1.34	3.10	3.36

>**3x** faster than FlashDecoding at prefill AND generation!

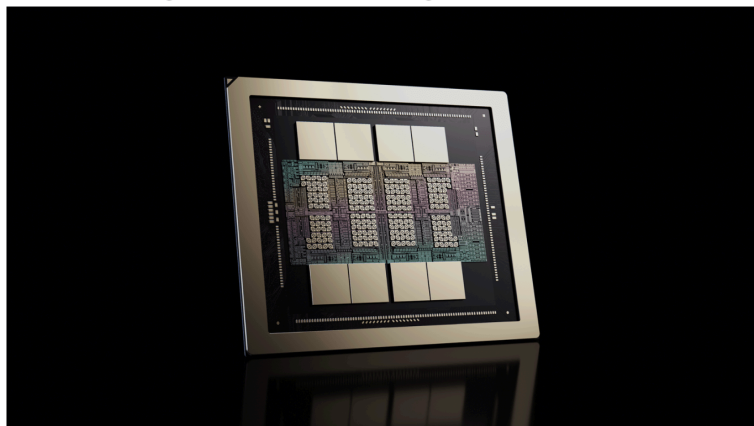
Final Thoughts

- On par with softmax on small sequences, (*much*) better on long context
- Triton, not CuteDSL/Cutile/CUDA ---> much room for improvements!
- Academic budget ---> We are trying to scale to larger models



Data Center / Cloud

Inside NVIDIA Rubin GPU Architecture: Powering the Era of Agentic AI



Jul 21, 2026

+32 Like Discuss (0)

By Eduardo Alvarez, Vishal Mehta and Farshad Ghodsian

GPUs + Dynamic Sparsity

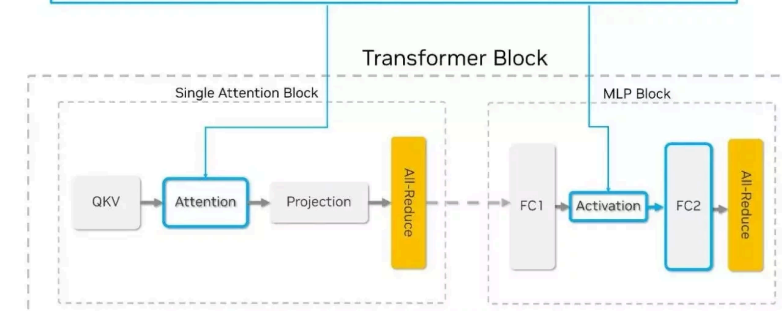
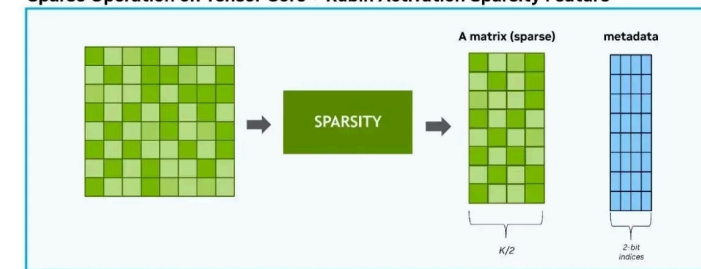
=



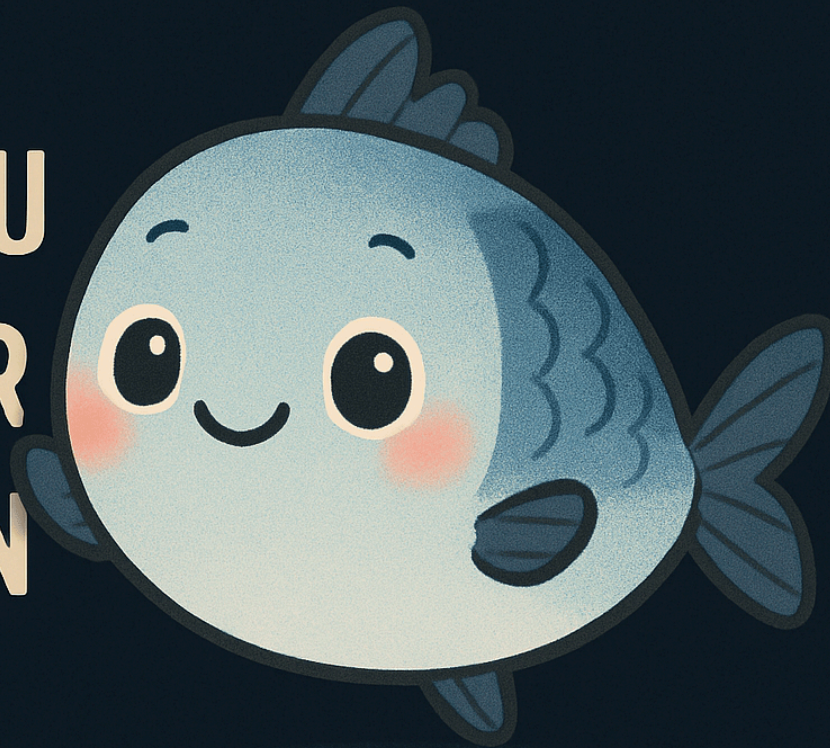
Rubin Adaptive Compression Sparsity

Rubin's sparsity feature – for use between attention related outputs

Sparse Operation on Tensor Core + Rubin Activation Sparsity Feature



**THANK YOU
FOR YOUR
ATTENTION**



Perguntas?