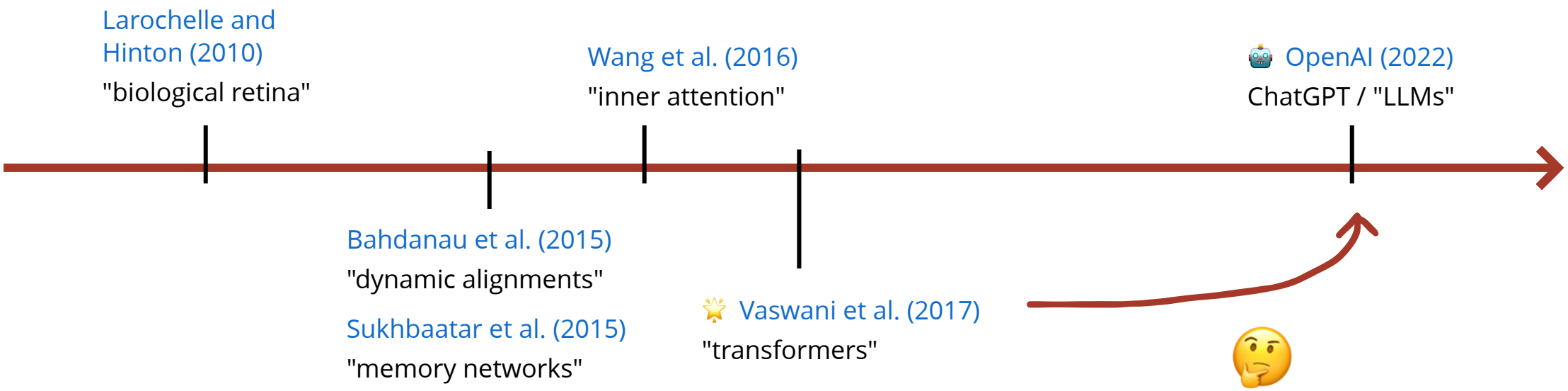
Pushing the Limits of Sparse Attention in LLMs

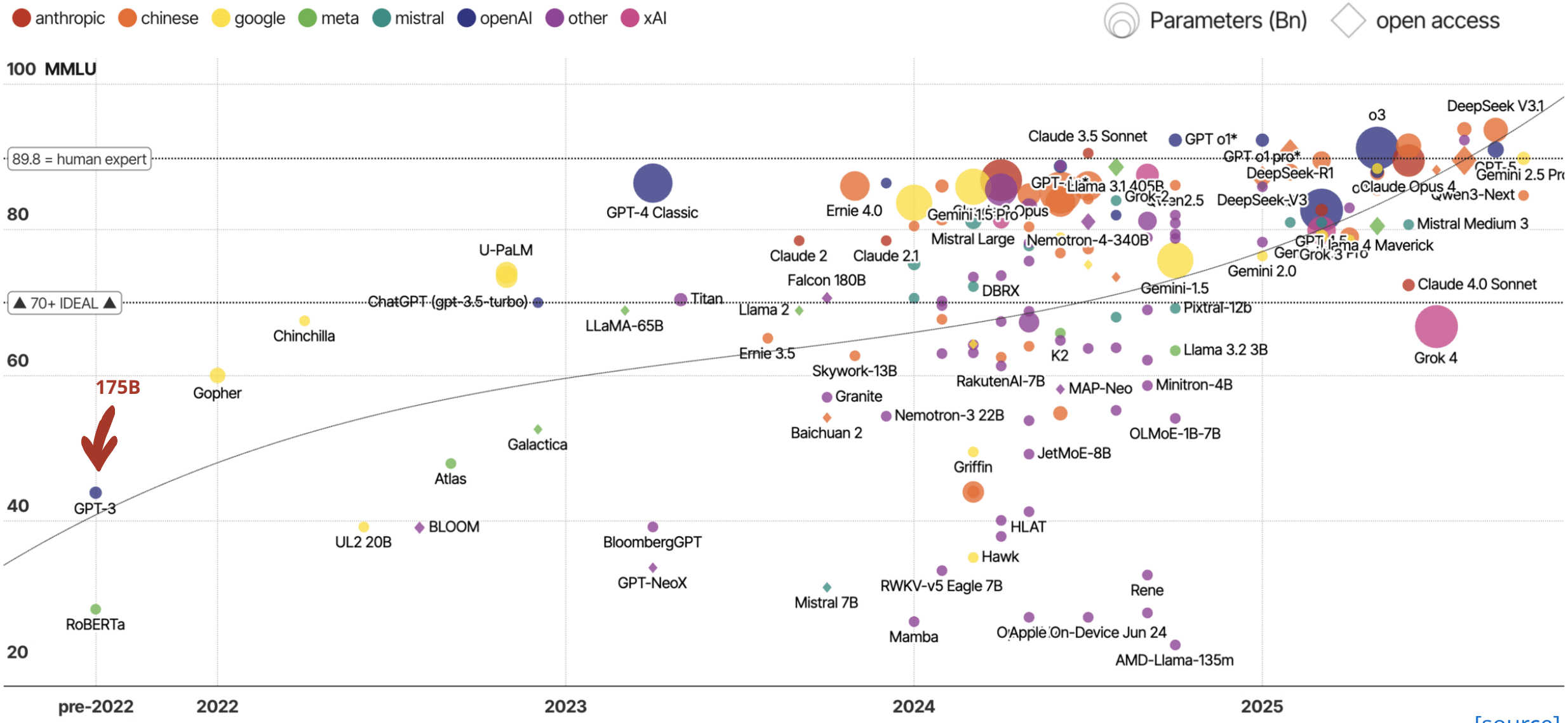
Marcos V. Treviso
Assist. Professor at IST / Univ. of Lisbon

ASAP Seminar

Large Language Models (LLMs)

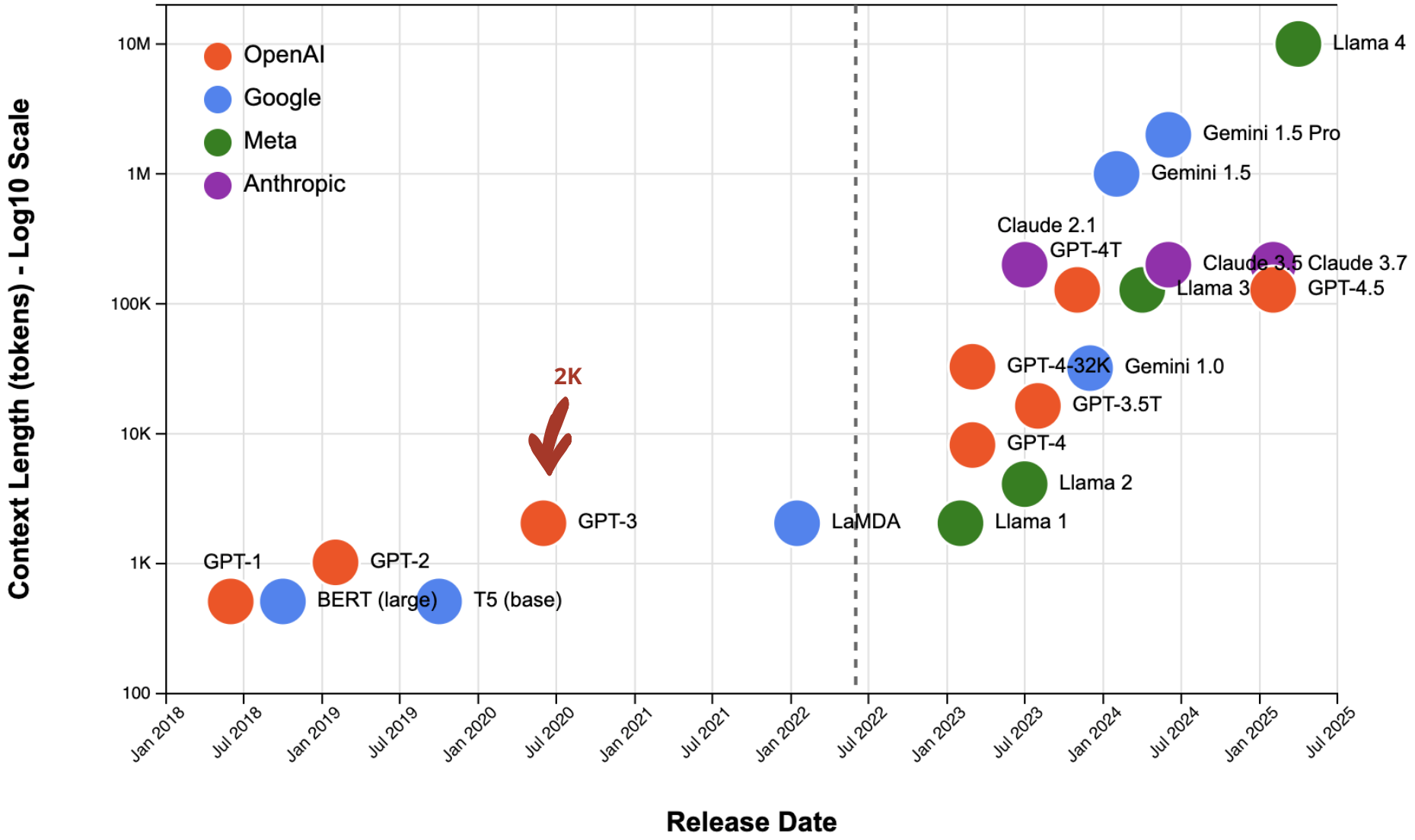


Major LLMs

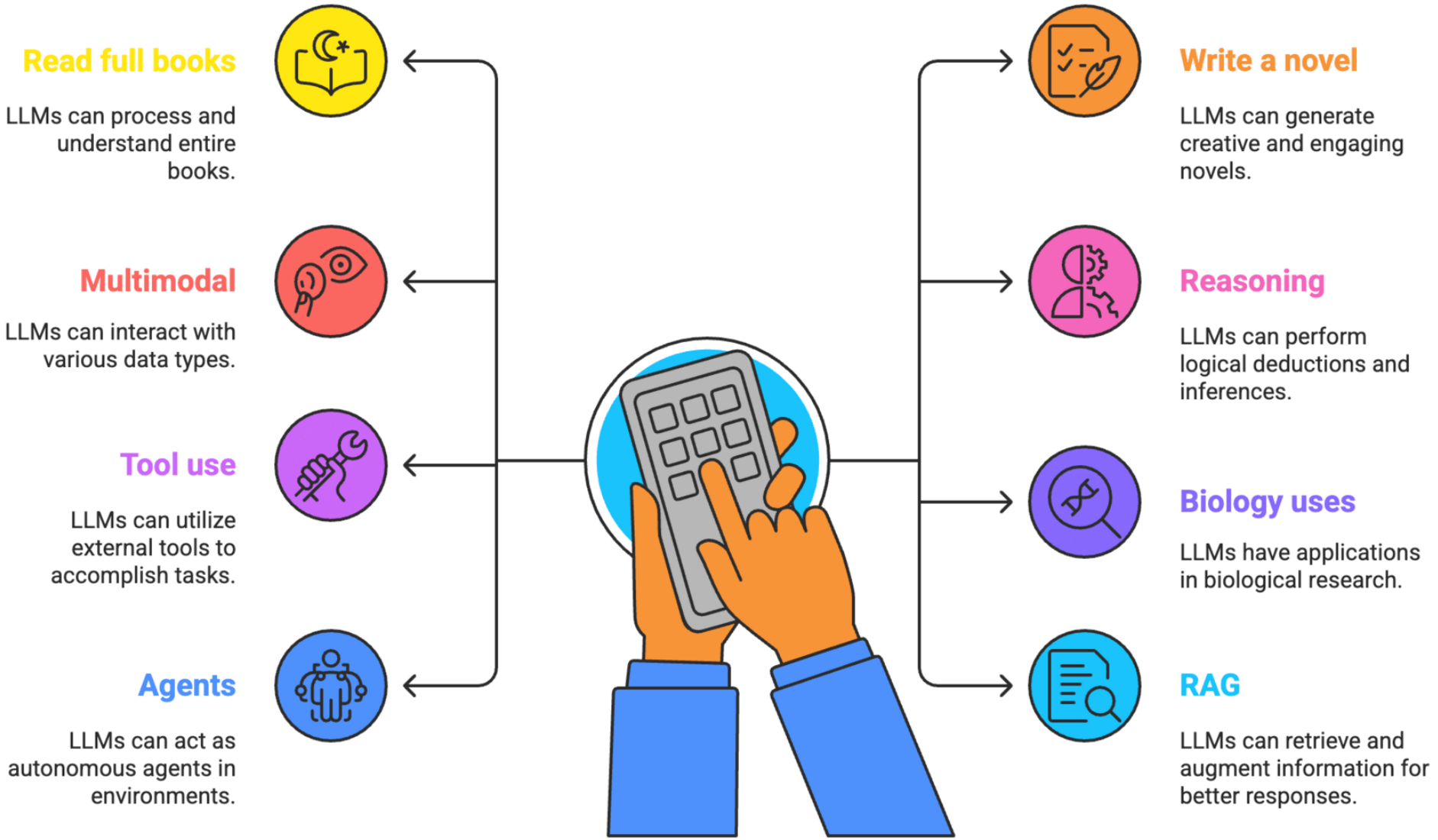


[source]

Context Length++



Capabilities



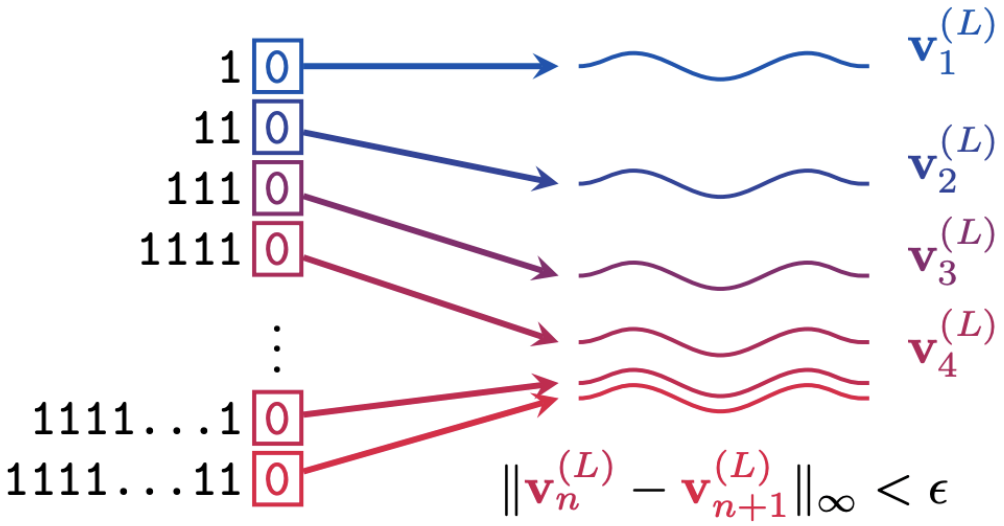
Limitations

What happens as we increase the context length? Can we just keep increasing?

Limitations

What happens as we increase the context length? Can we just keep increasing?

Representational Collapse

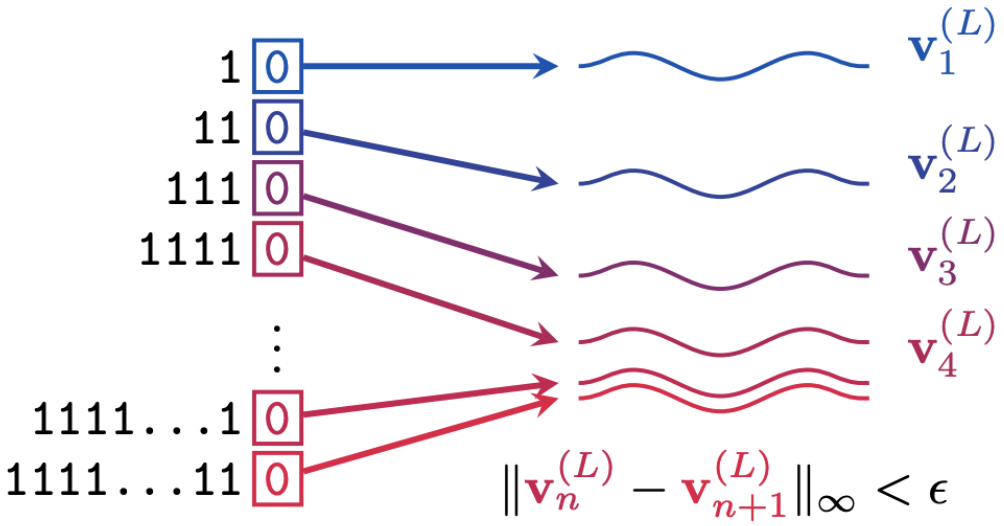


As we add context tokens, the final representation of "0" collapse to a single vector

Limitations

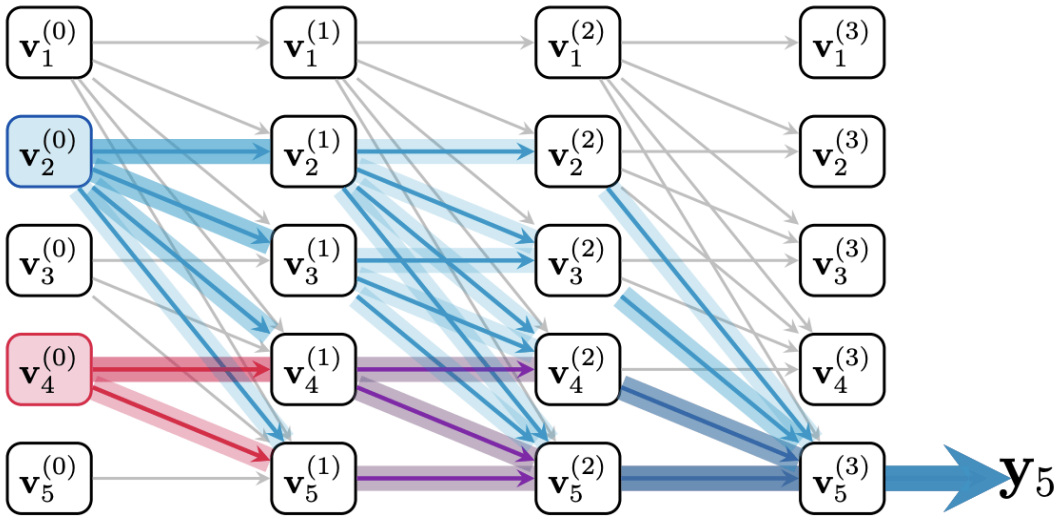
What happens as we increase the context length? Can we just keep increasing?

Representational Collapse



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

Oversquashing

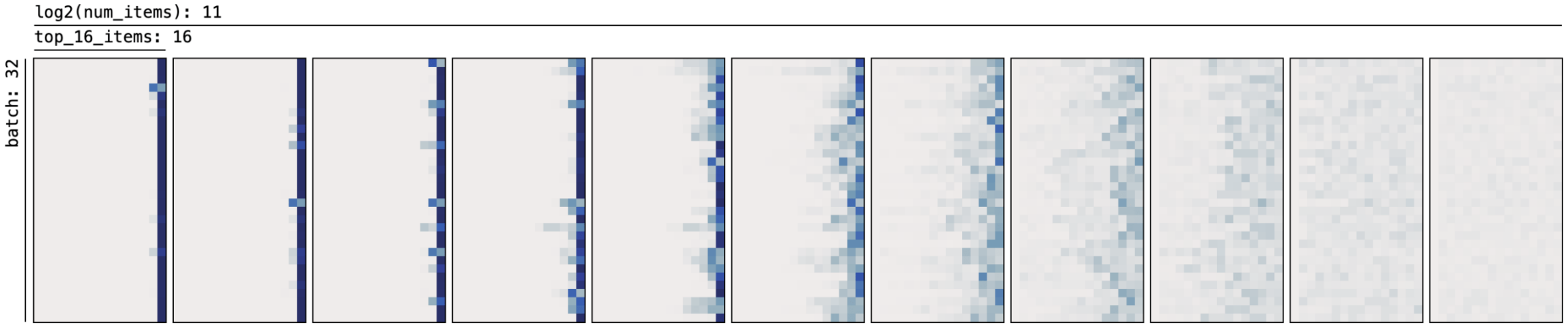


A lot of information from context tokens is forced through a single, fixed-size bottleneck

Limitations

What happens as we increase the context length? Can we just keep increasing it?

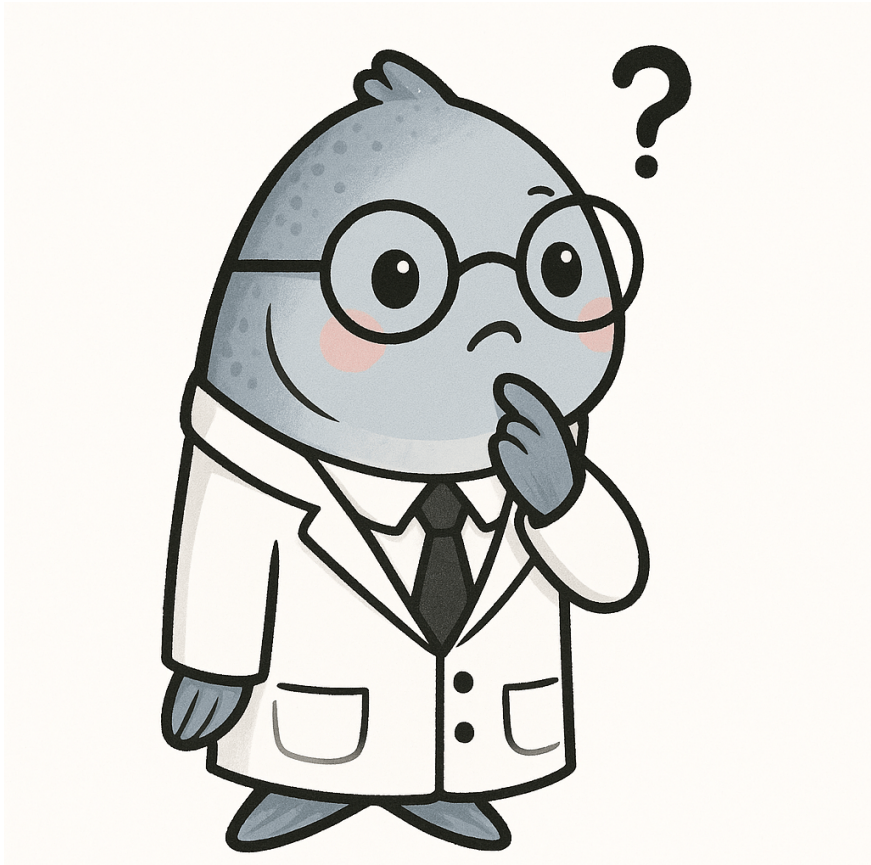
Attention Dispersion



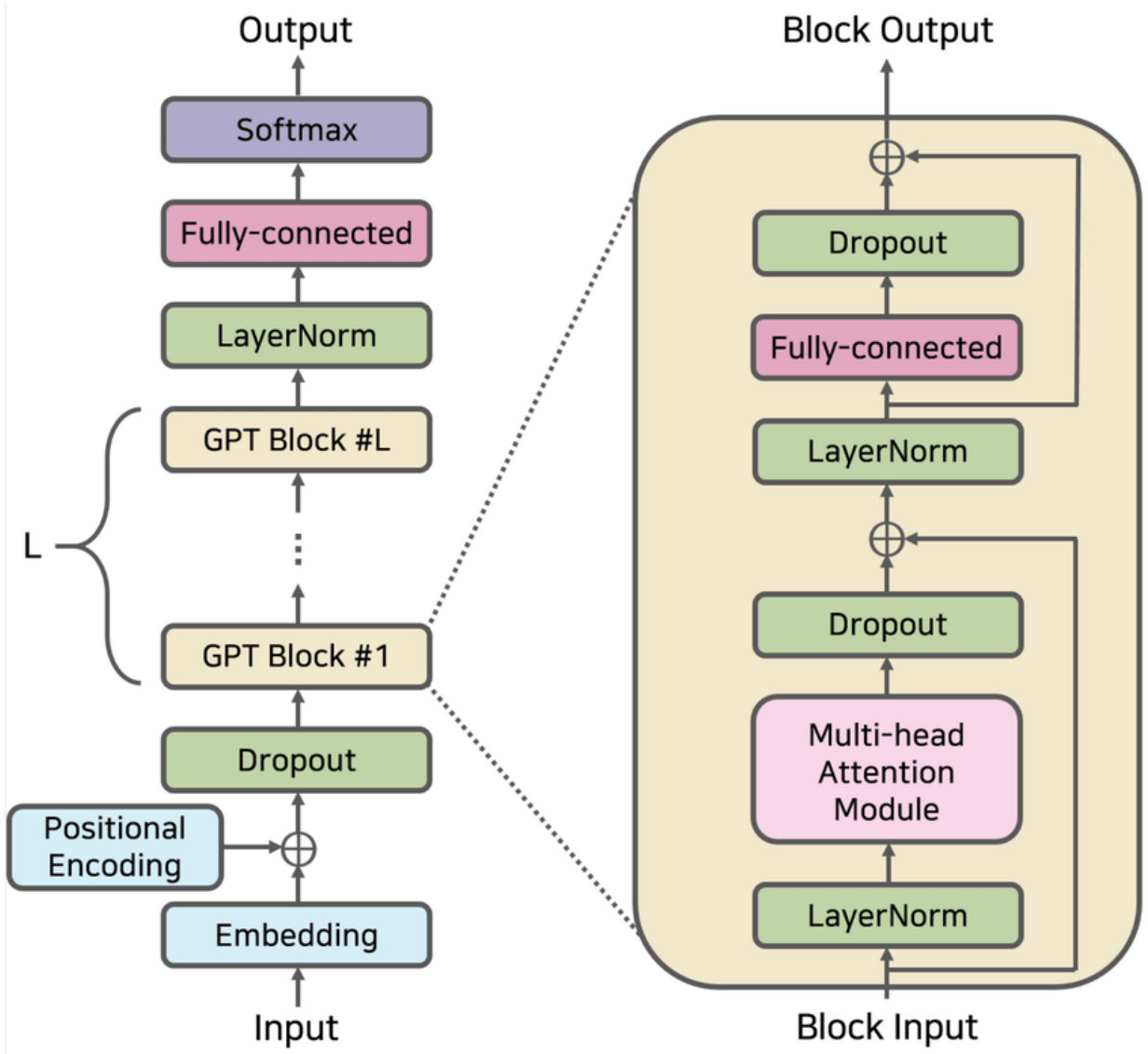
As the number of tokens increase, the attention coefficients spread out (uniform distribution)

~~`output = softmax(q @ k.t()) @ v`~~ `output = v.mean(1)`

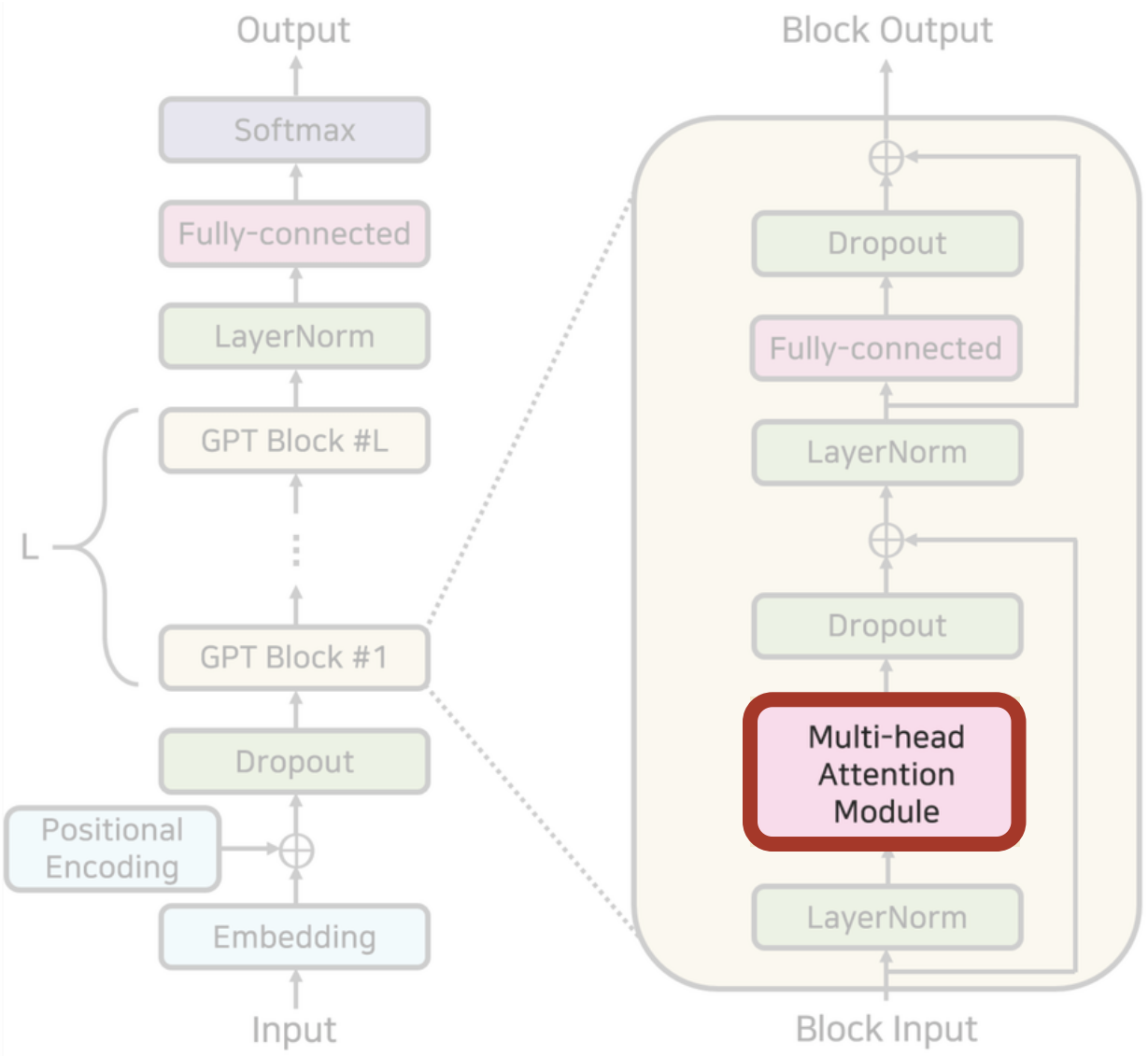
What is going on?



Generative *Transformers*



Generative *Transformers*



Self-Attention

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

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

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

π maps scores to probabilities (row-wise)

$$\begin{cases} \boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n} \\ \boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n} \\ \boldsymbol{O} = \boldsymbol{P}\boldsymbol{V} \in \mathbb{R}^{n \times d} \end{cases}$$

Self-Attention

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

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

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

π maps scores to probabilities (row-wise)

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

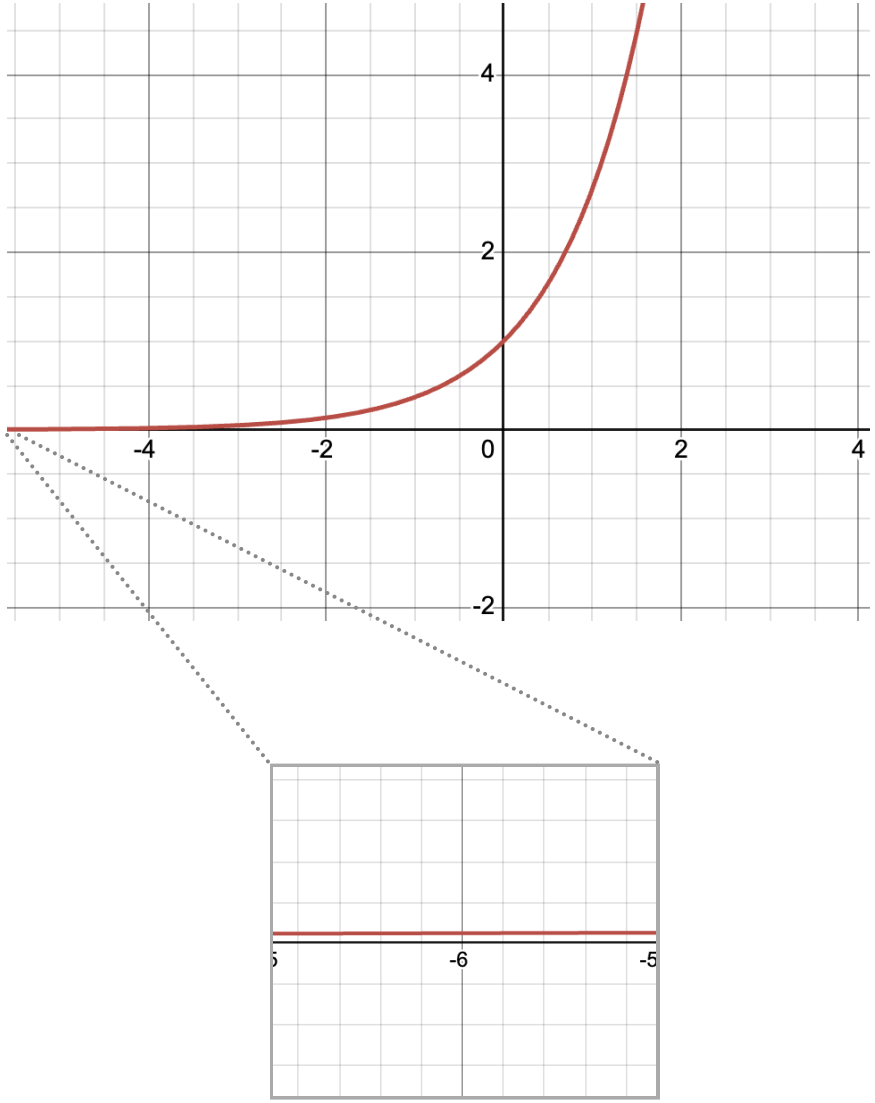
$$\begin{cases} \boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n} \\ \boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n} \\ \boldsymbol{O} = \boldsymbol{P}\boldsymbol{V} \in \mathbb{R}^{n \times d} \end{cases}$$

The culprit? Softmax

But what exactly is softmax?

$$\text{softmax}(\boldsymbol{z})_j = \frac{\exp(\boldsymbol{z}_j)}{\sum_k \exp(\boldsymbol{z}_k)}$$

- The exp function is always positive (> 0)
  - All probabilities are positive
  - All "computation paths" are active
  - All "tokens" contribute
  - **No focus at extreme lengths**

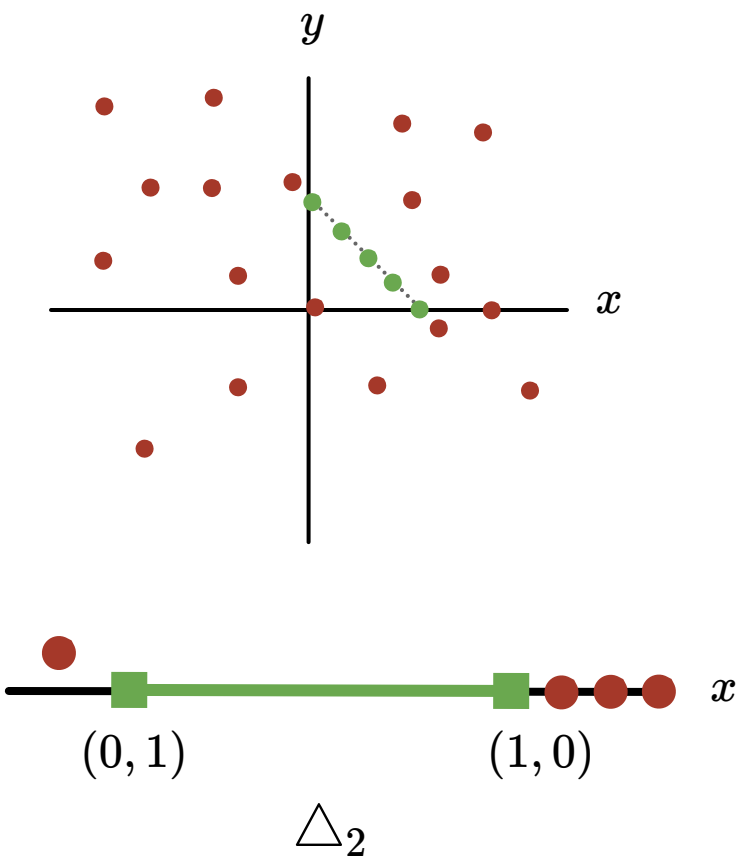


Softmax origins: probability simplex

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

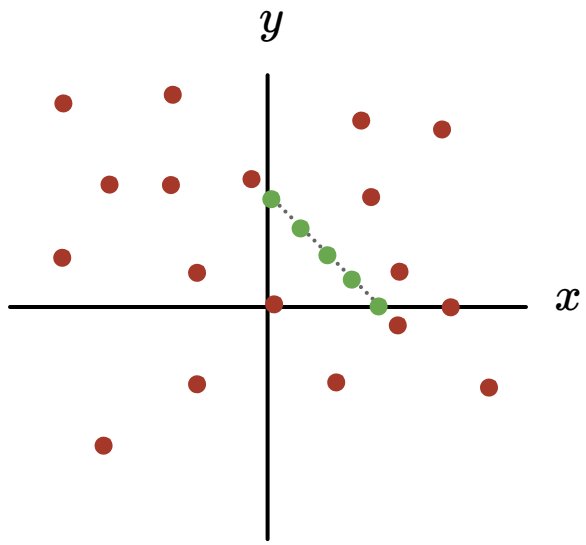
Softmax origins: probability simplex

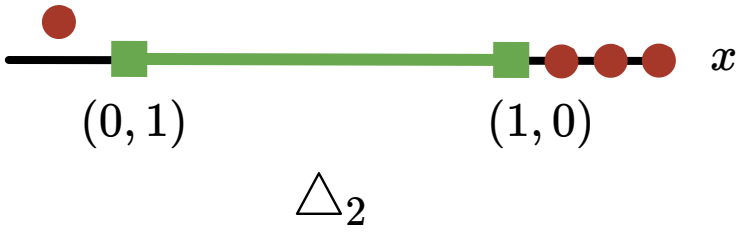
$$\triangle_n = \{\boldsymbol{p} \in \mathbb{R}_+^n \mid \sum_i p_i = 1\}$$



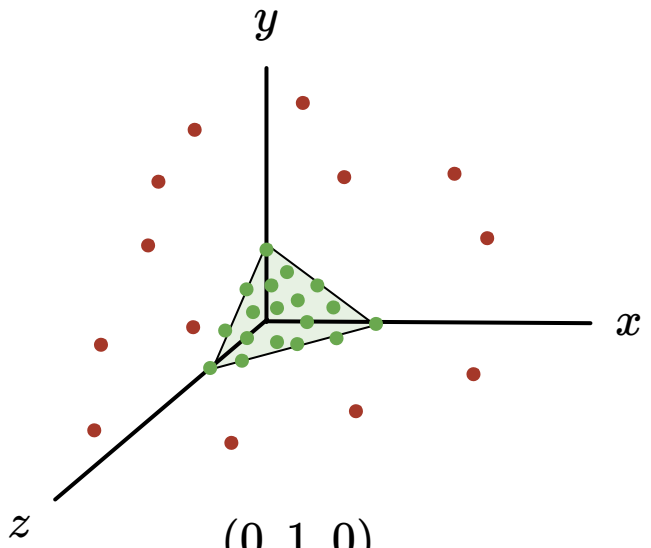
Softmax origins: probability simplex

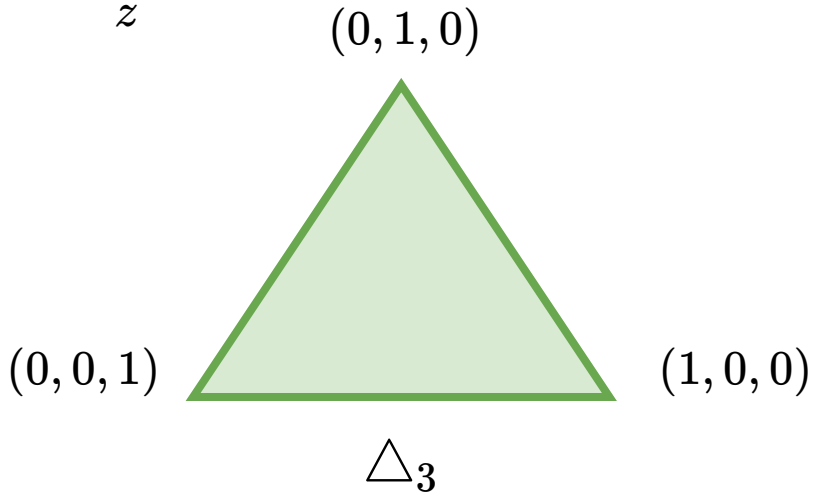
$$\triangle_n = \{\boldsymbol{p} \in \mathbb{R}^n_+ \mid \sum_i p_i = 1\}$$





$\triangle_2$

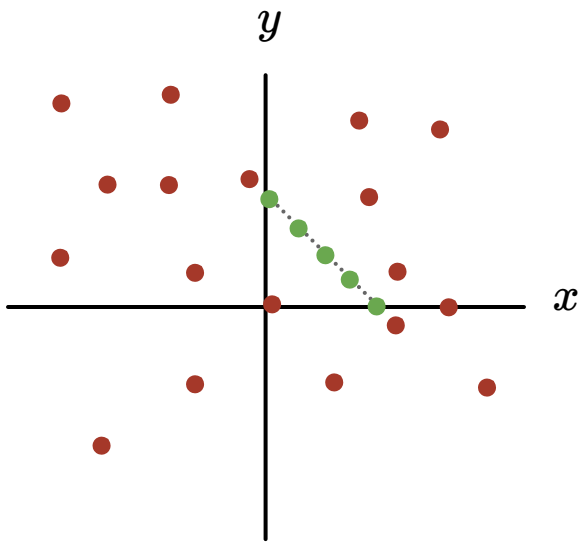


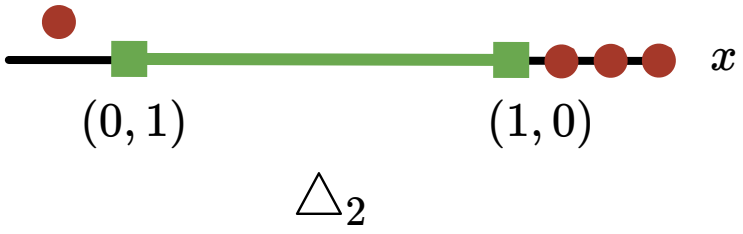


$\triangle_3$

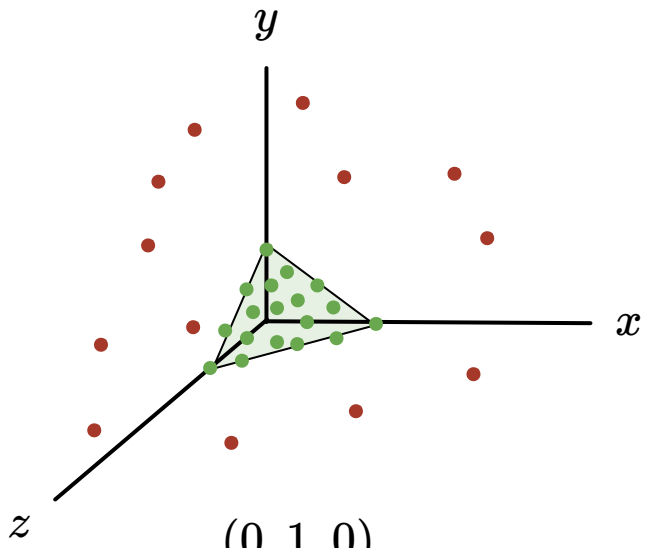
Softmax origins: probability simplex

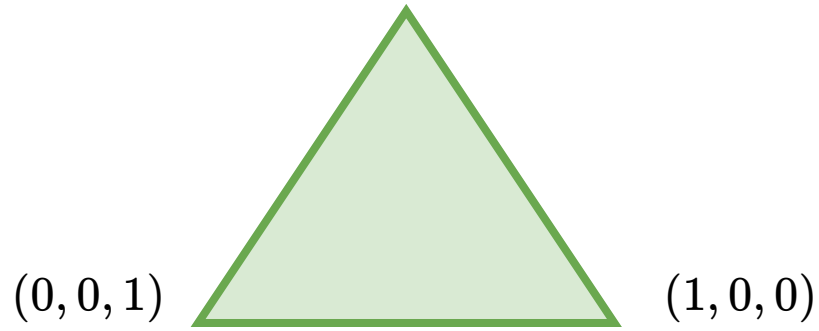
$$\triangle_n = \{\boldsymbol{p} \in \mathbb{R}^n_+ \mid \textstyle\sum_i p_i = 1\}$$





$\triangle_2$



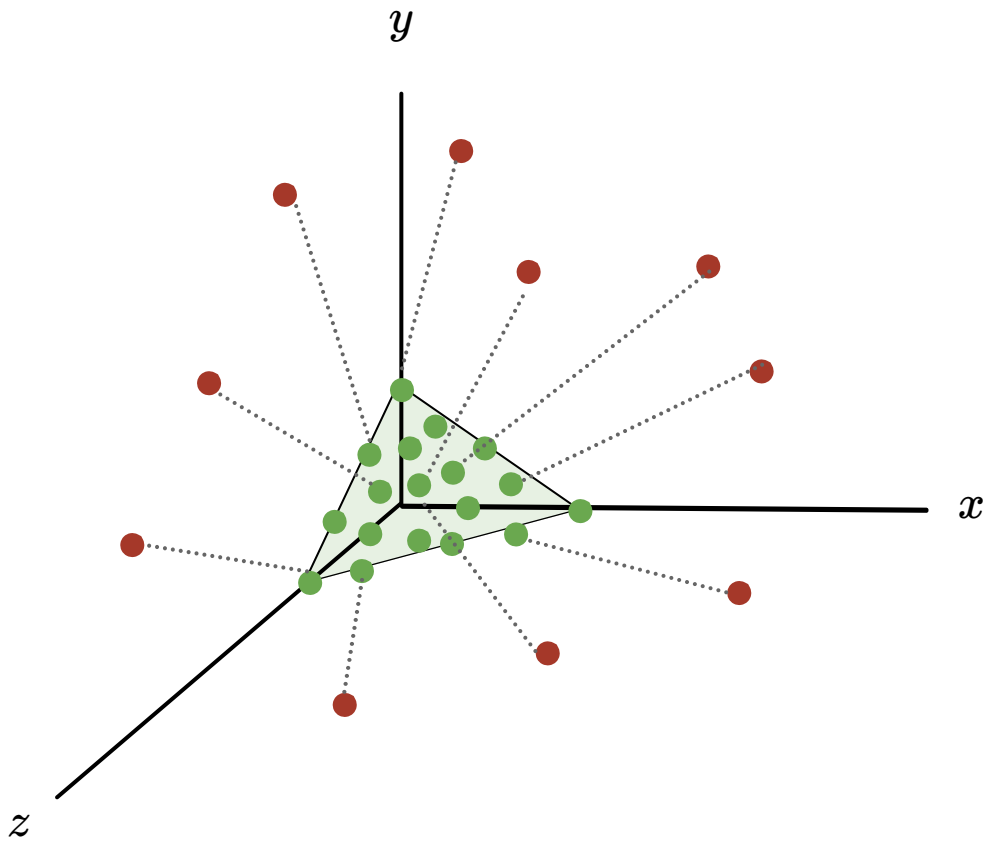


$\triangle_3$

$\triangle_4$

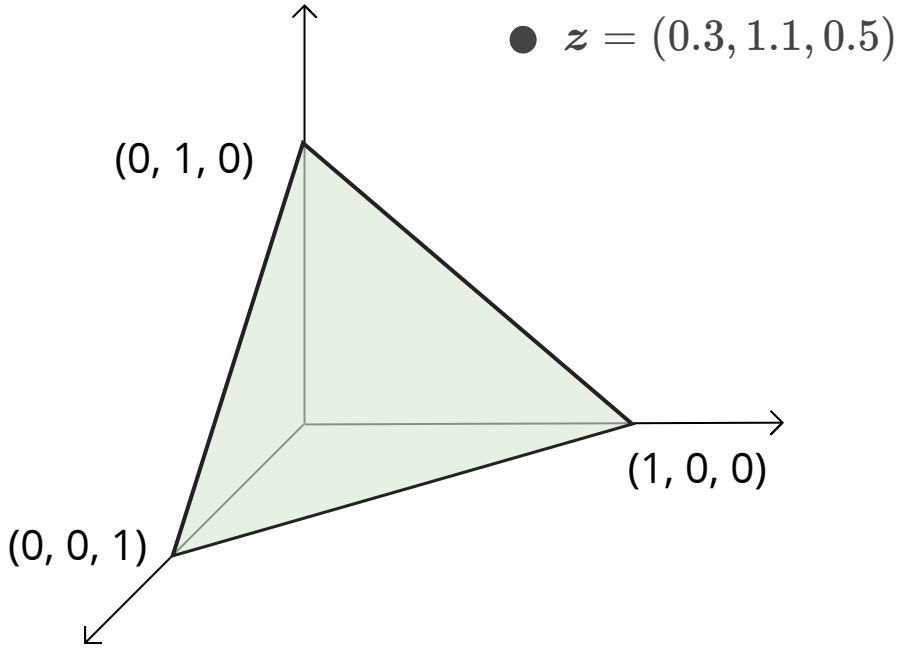
How can we map scores to probabilities?

Easy! Map to the probability simplex!



How can we map scores to probabilities?

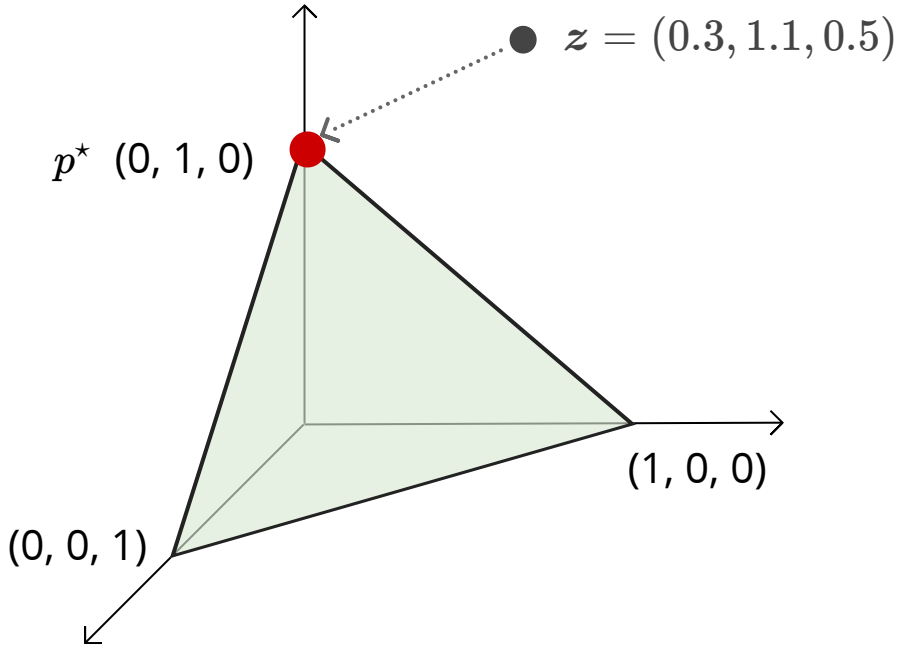
$$\max \boldsymbol{z} = \max_{\boldsymbol{p} \in \triangle_n} \boldsymbol{p}^\top \boldsymbol{z}$$



Fundamental Theorem of Linear Programming
(Dantzig et al., 1955)

How can we map scores to probabilities?

$$\max \boldsymbol{z} = \max_{\boldsymbol{p} \in \triangle_n} \boldsymbol{p}^\top \boldsymbol{z} \quad \longrightarrow \quad \boldsymbol{p}^\star = \arg\max_{\boldsymbol{p} \in \triangle_n} \boldsymbol{p}^\top \boldsymbol{z}$$



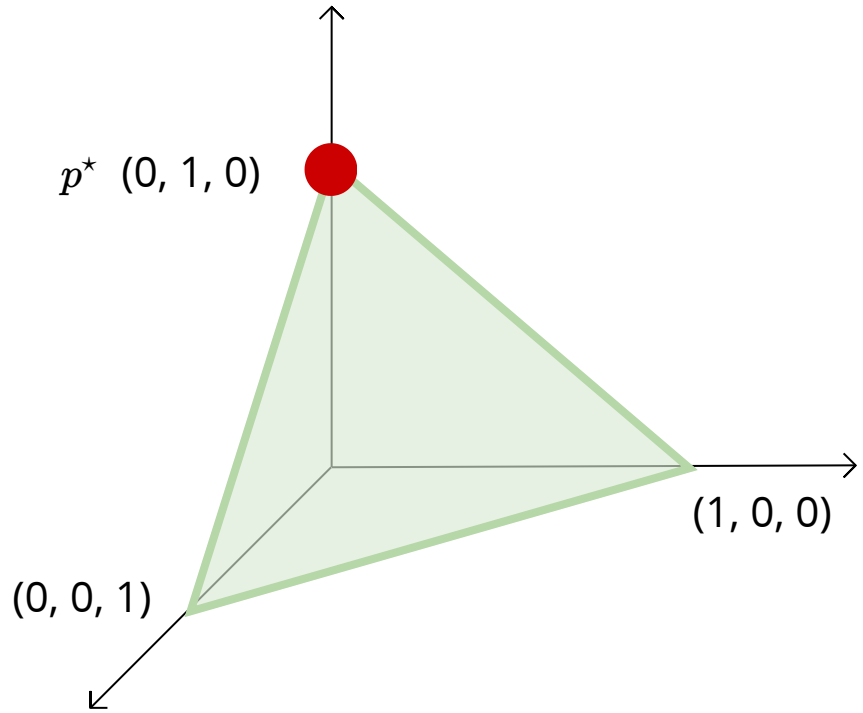
Fundamental Theorem of Linear Programming
(Dantzig et al., 1955)

How can we map scores to probabilities?

Entropy regularizer

$$\boldsymbol{p}^\star = \underset{\boldsymbol{p} \in \triangle_n}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{z} - \Omega(\boldsymbol{p})$$

- argmax: $\Omega(\boldsymbol{p}) = 0$



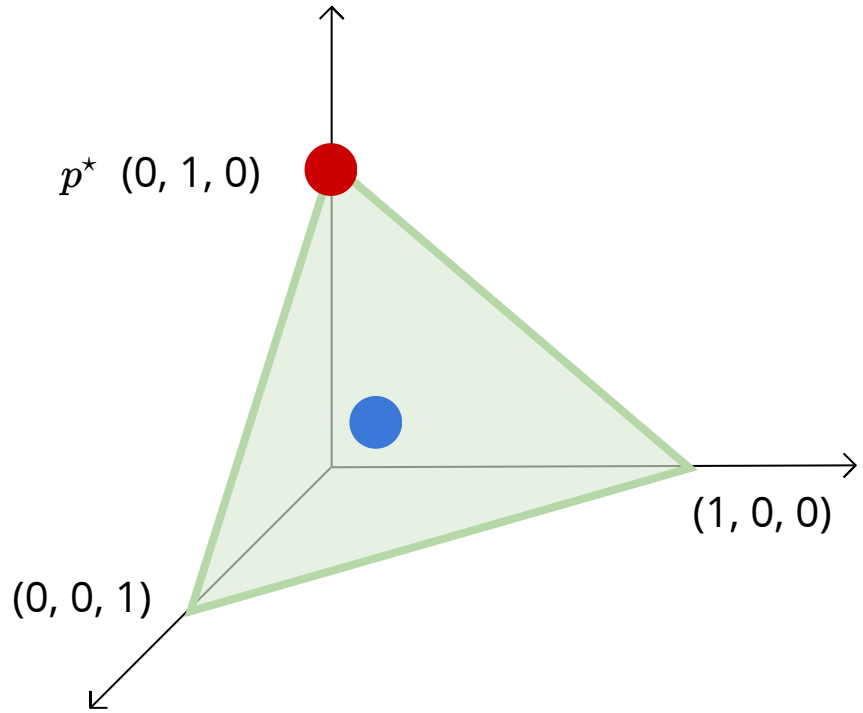
How can we map scores to probabilities?

Entropy regularizer

$$\boldsymbol{p}^\star = \underset{\boldsymbol{p} \in \triangle_n}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{z} - \Omega(\boldsymbol{p})$$

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

(it should be called "softargmax")

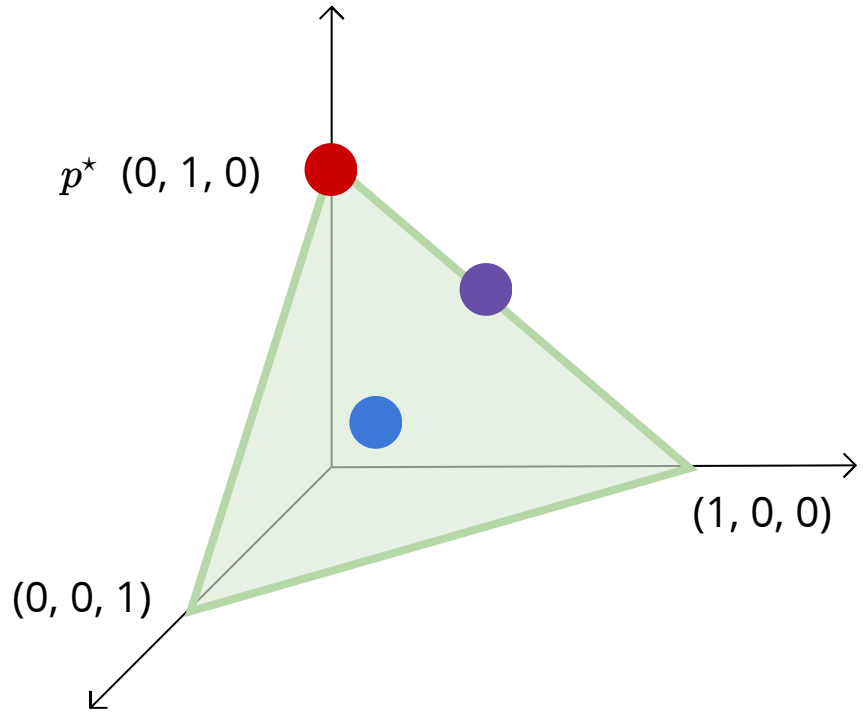


How can we map scores to probabilities?

Entropy regularizer

$$\boldsymbol{p}^\star = \underset{\boldsymbol{p} \in \triangle_n}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{z} - \Omega(\boldsymbol{p})$$

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



(Martins and Astudillo, 2016)

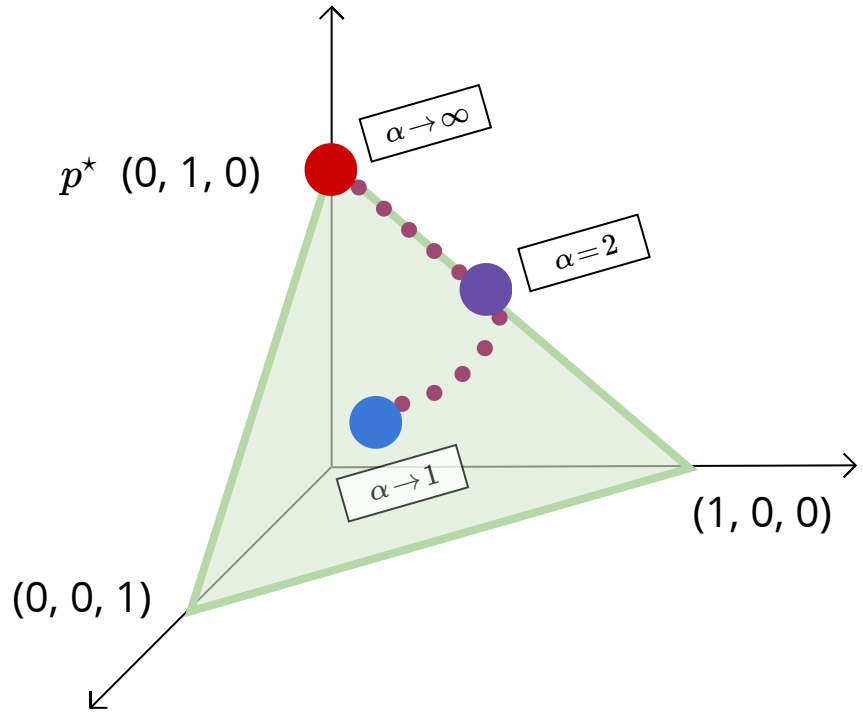
How can we map scores to probabilities?

Entropy regularizer

$$\boldsymbol{p}^\star = \underset{\boldsymbol{p} \in \triangle_n}{\arg\max}\, \boldsymbol{p}^\top \boldsymbol{z} - \Omega(\boldsymbol{p})$$

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

{ softmax when $\alpha \to 1$
  sparsemax when $\alpha = 2$



(Peters, Niculae, and Martins, 2019)

Computing α-entmax

$$\boldsymbol{p} = \exp(\boldsymbol{z}) / \sum_k \exp(\boldsymbol{z})_k$$

```
def softmax(z):
  return z.exp() / z.exp().sum()
```

Computing α-entmax

$$\boldsymbol{p} = \exp(\boldsymbol{z}) / \sum_k \exp(\boldsymbol{z})_k$$

```
def softmax(z):
  return z.exp() / z.exp().sum()
```

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ can also be evaluated easily:

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

```
def sparsemax(z, tau):
  return max(0, z - tau)
```

Computing α-entmax

$$\boldsymbol{p} = \exp(\boldsymbol{z}) / \sum_k \exp(\boldsymbol{z})_k$$

```
def softmax(z):
  return z.exp() / z.exp().sum()
```

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ can also be evaluated easily:

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

for $\alpha = 2$ (sparsemax)

$$\boldsymbol{p} = \max(0, \boldsymbol{z} - \tau)$$

```
def sparsemax(z, tau):
  return max(0, z - tau)
```

such that $\sum_i p_i = 1$

a constant that ensures the whole expression sums to 1

Computing α-entmax

$$\boldsymbol{p} = \exp(\boldsymbol{z}) / \sum_k \exp(\boldsymbol{z})_k$$

```
def softmax(z):
  return z.exp() / z.exp().sum()
```

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ can also be evaluated easily:

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

for $\alpha = 2$ (sparsemax)

$$\boldsymbol{p} = \max(0, \boldsymbol{z} - \tau)$$

such that $\sum_i p_i = 1$

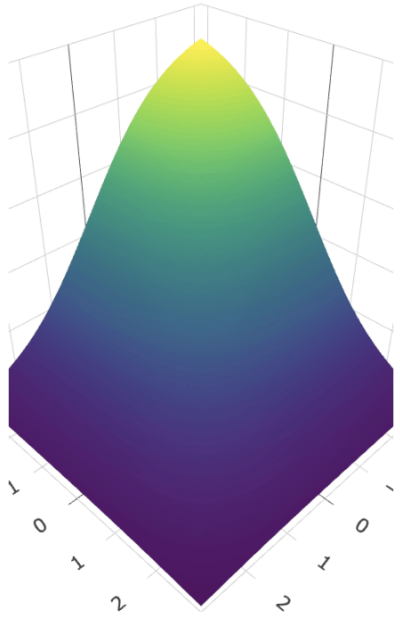
a constant that ensures the whole expression sums to 1

```
def sparsemax(z, tau):
  return max(0, z - tau)
```

😢 Finding the optimal $\tau \in \mathbb{R}$ boils down to solving an optimization problem...

🤔 Why complicate so much? If we want sparsity, why not just doing top-k?

softmax



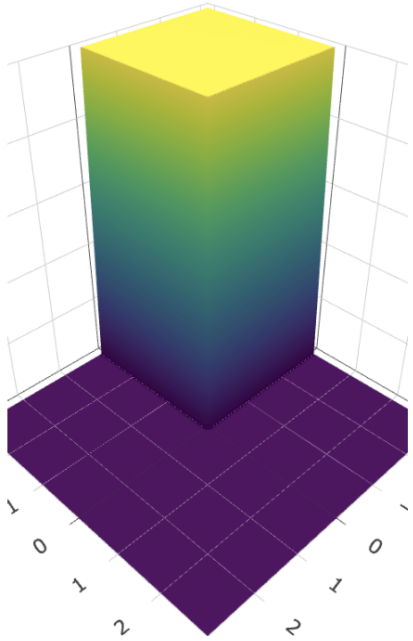
Easy to evaluate

GPU-friendly

Differentiable

Dense

top-k



Slow to evaluate

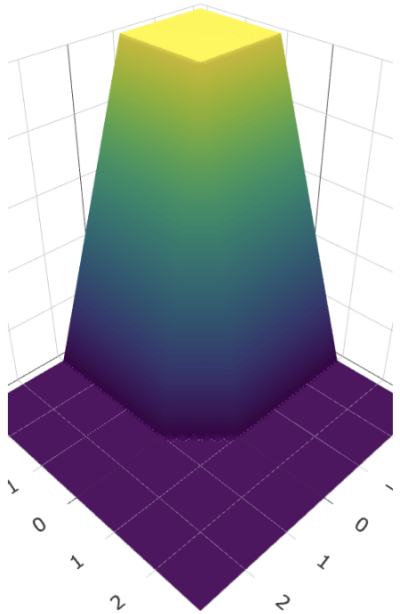
Not GPU-friendly

Non-differentiable

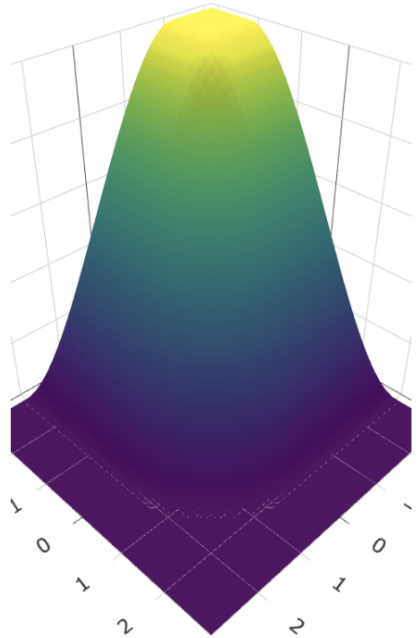
Sparse

α-entmax

sparsemax ($\alpha = 2$)



1.5-entmax



Easy to evaluate

GPU-friendly*

Differentiable

Sparse

Long-context Generalization with Sparse Attention


Pavlo Vasylenko


Hugo Pitorro


André F. T. Martins


Marcos Treviso

What happens with α-entmax attention on long sequences?

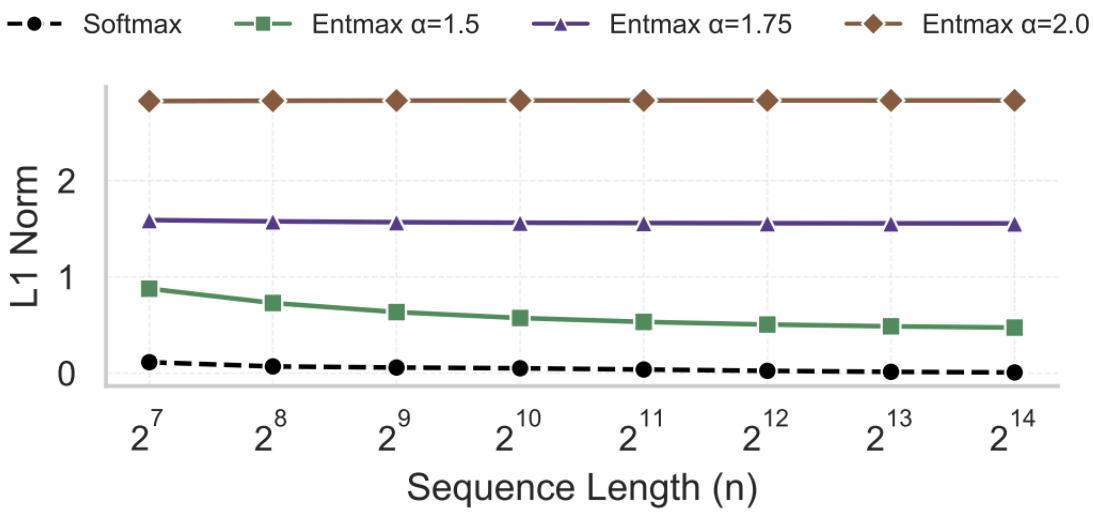
📄 arxiv.org/abs/2506.16640

Theoretical Findings

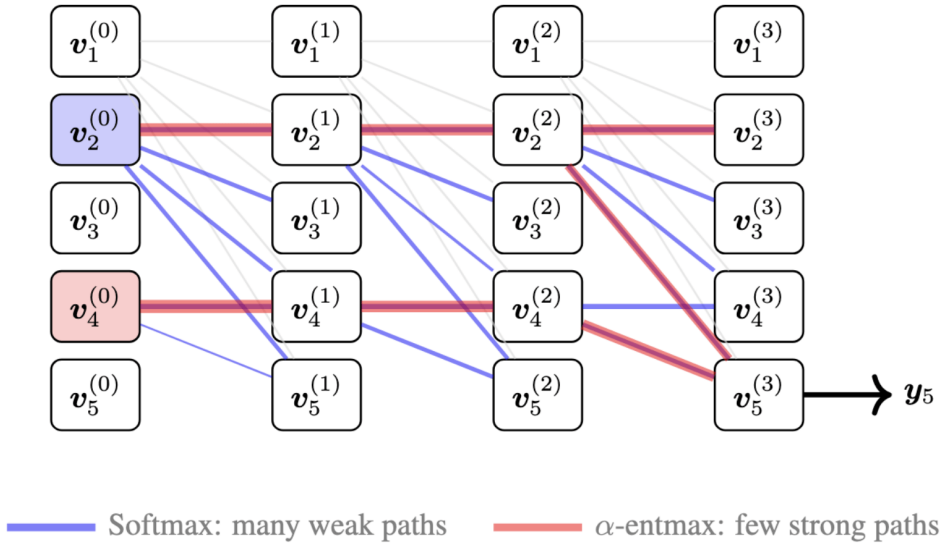
Proposition 2 (Representational Preservation and Reduced Gradient Paths)**.** *Let $\alpha > 1$. Consider a depth-L transformer with residual connections and attention weights given by α-entmax. Suppose each attention distribution has support size at most s with $s \ll n$. Then:*

1. ***(Preserved representations)*** *There exist input families $\boldsymbol{v}^{(0)} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{v}^{*(0)} \in \mathbb{R}^{(n+1) \times d}$ and a constant $c > 0$ such that $\|\boldsymbol{v}_n^{(L)} - \boldsymbol{v}_{n+1}^{*(L)}\|_1 \geq c$ for all n; i.e., α-entmax can maintain distinct token representations as $n \to \infty$.*
2. ***(Alleviated over-squashing)*** *The number of effective gradient paths scales as $\mathcal{O}(s^L)$ (rather than $\mathcal{O}(n^L)$ under softmax), alleviating over-squashing via stronger gradient signals.*

✅ Preserve representations



✅ Alleviate over-squashing



Theoretical Findings

Proposition 1 (Dispersion Properties of Attention Mechanisms). *Comparing softmax and α-entmax ($\alpha > 1$) attention mechanisms:*

1. ***α-entmax can retain probability, while softmax always leaks:*** *For any $\alpha > 1$ and any logits $\boldsymbol{z} \in \mathbb{R}^n$, there are logits $\boldsymbol{z}^* \in \mathbb{R}^N$ with $N > n$ such that:*

$$\alpha\text{-entmax}(\boldsymbol{z})_i = \alpha\text{-entmax}(\boldsymbol{z}^*)_i \quad \forall i \leq n. \tag{5}$$

   This is impossible for $\alpha = 1$ (softmax), for which we always have $\text{softmax}(\boldsymbol{z})_i < \text{softmax}(\boldsymbol{z}^*)_i$.

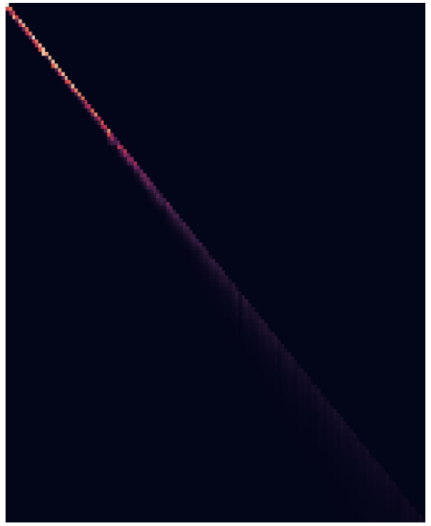
2. ***Softmax exhibits complete dispersion:*** *For any fixed temperature $\theta > 0$ and any bounded sequence of logits $(z_n)_{n \in \mathbb{N}}$:*

$$\lim_{n \to \infty} \frac{H(\text{softmax}(\boldsymbol{z}_{1:n}/\theta))}{\log n} = 1. \tag{6}$$

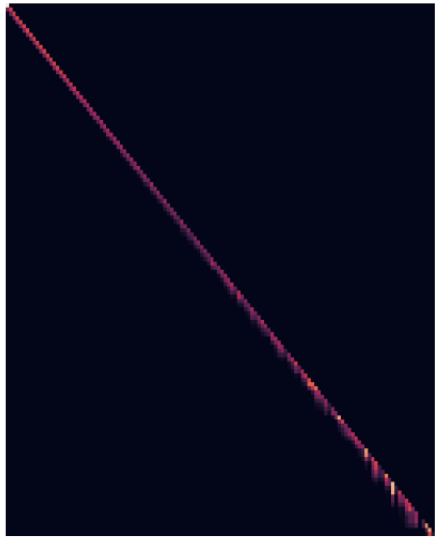
3. ***α-entmax can exhibit strong concentration resilience:*** *When the support size grows sublinearly as $|\mathcal{S}| = \mathcal{O}(n^\beta)$ with $\beta < 1$, α-entmax maintains bounded normalized entropy:*

$$\lim_{n \to \infty} \frac{H(\alpha\text{-entmax}(\boldsymbol{z}_{1:n}))}{\log n} \leq \beta < 1. \tag{7}$$

✅ Maintain focus (non-dispersion)



Softmax



Entmax

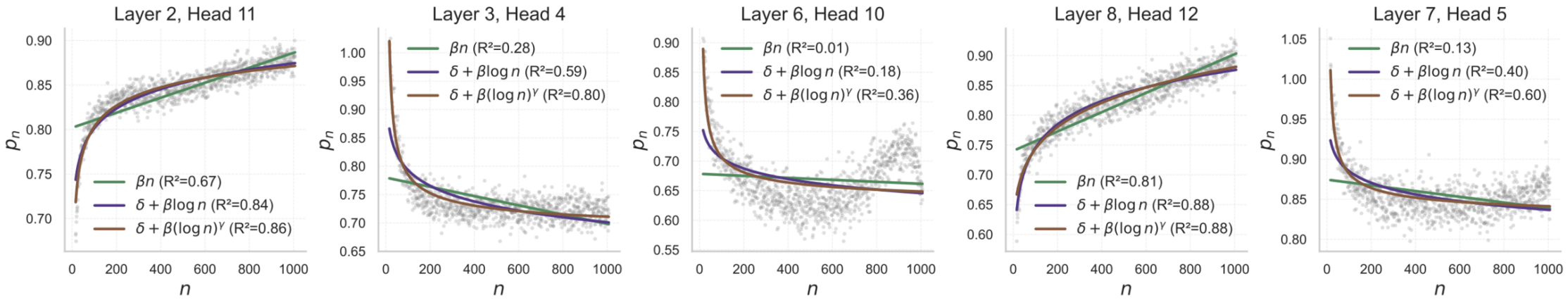
Adaptive-Scalable α-entmax (ASEntmax)

What if entmax ignores *too many* relevant tokens?

$$\text{ASEntmax}(\boldsymbol{Z}) = \alpha\text{-entmax}((\underline{\delta + \beta(\log n)^{\gamma}})\boldsymbol{z})$$

scaling to counteract excessive sparsification

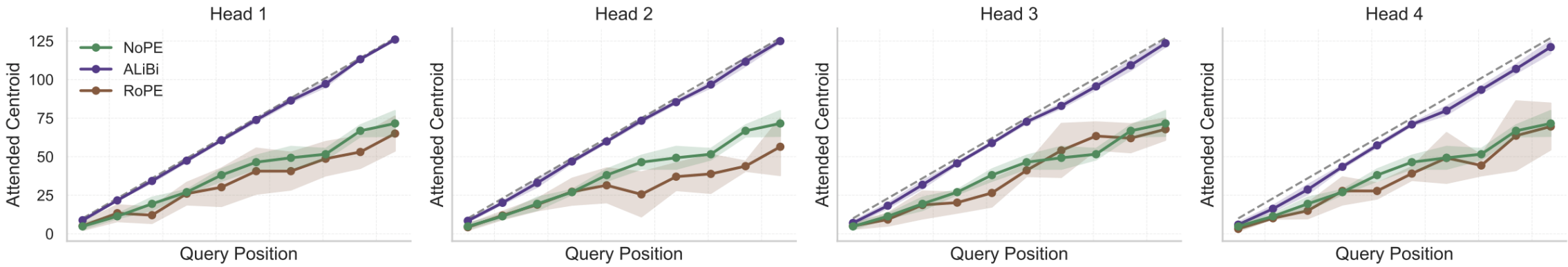
δ, β, γ are *learnable* for each ***head*** and ***query***.



Positional Encoding for Sparse Attention

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

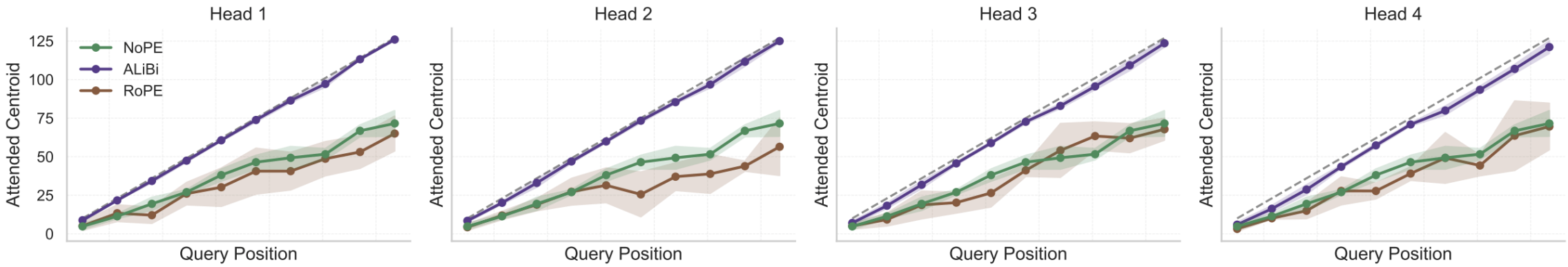
$\Longrightarrow$ hard cutoffs and "dead zones" that break long-range links



Positional Encoding for Sparse Attention

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

$\Longrightarrow$ hard cutoffs and "dead zones" that break long-range links



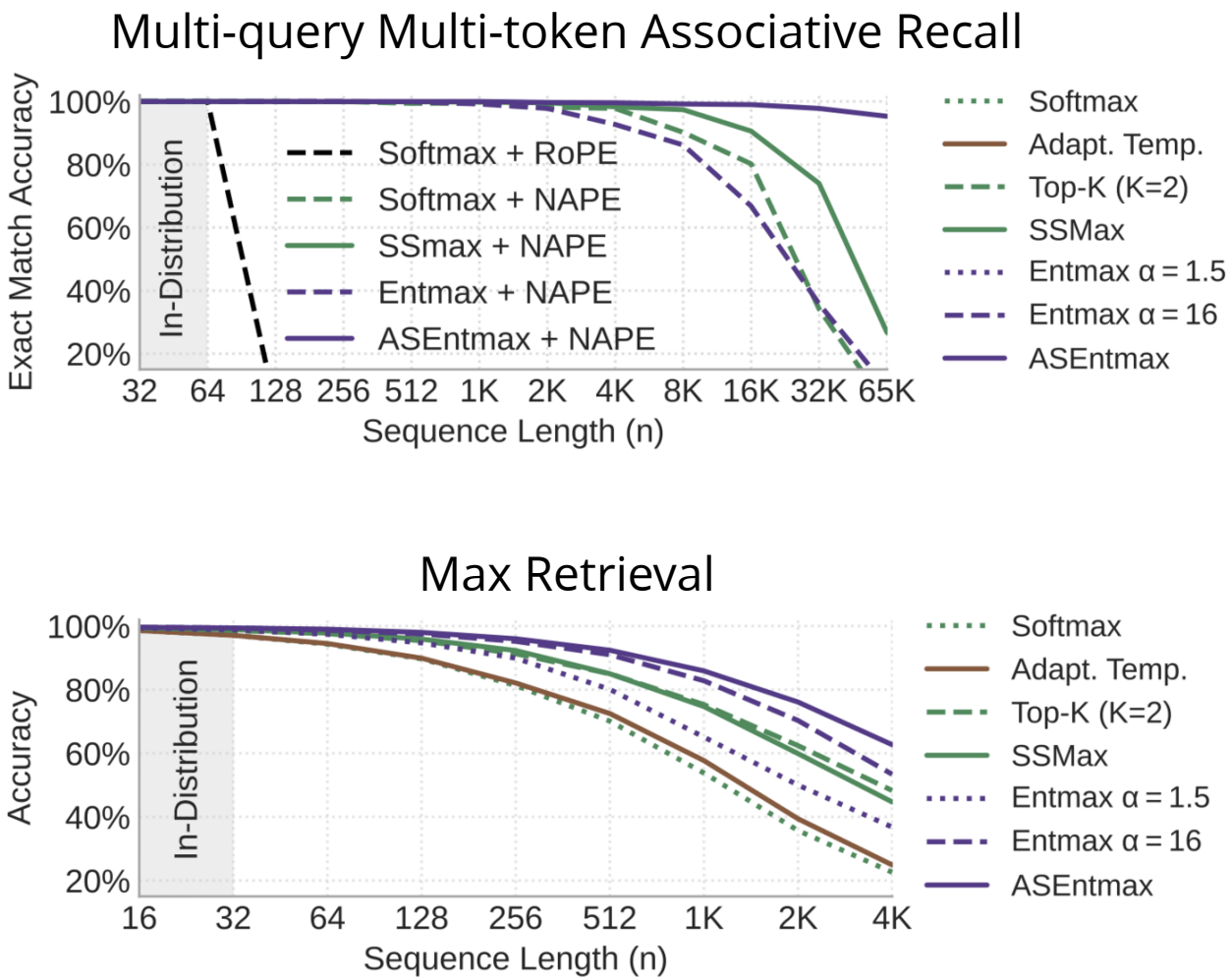
🌟 **NAPE (NoPE + ALiBi):** half of the heads use NoPE, the other half use ALiBi

- short-span focus

- semantic-guided attention

ps: also good for softmax

Synthetic tasks



copy, reverse, sort, 2-back, flip-flop in the paper with very similar trends

Language Modeling

- 420M Transformer++ trained from scratch on ~10B tokens of DCLM-Edu

Method	Lambada (PPL)	Lambada	Hellaswag	PIQA	Arc-C	WinoGrande	OpenbookQA
Softmax	52.4	30.9	33.1	**65.1**	25.6	49.5	28.2
SSMax	48.9	31.6	32.9	**65.1**	25.0	**51.5**	**30.4**
Entmax	47.9	32.1	32.8	63.6	24.6	50.9	29.0
ASEntmax	**41.6**	**34.3**	**33.4**	63.8	**26.0**	50.0	28.6

- 1B models show the same trend: ASEntmax outperforms on Lambada and on par on other datasets

Language Modeling

	ArXiv (ID)		ArXiv (OOD)			PubMed (ID)		PubMed (OOD)		
Model	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**

RULER – Needle in a haystack

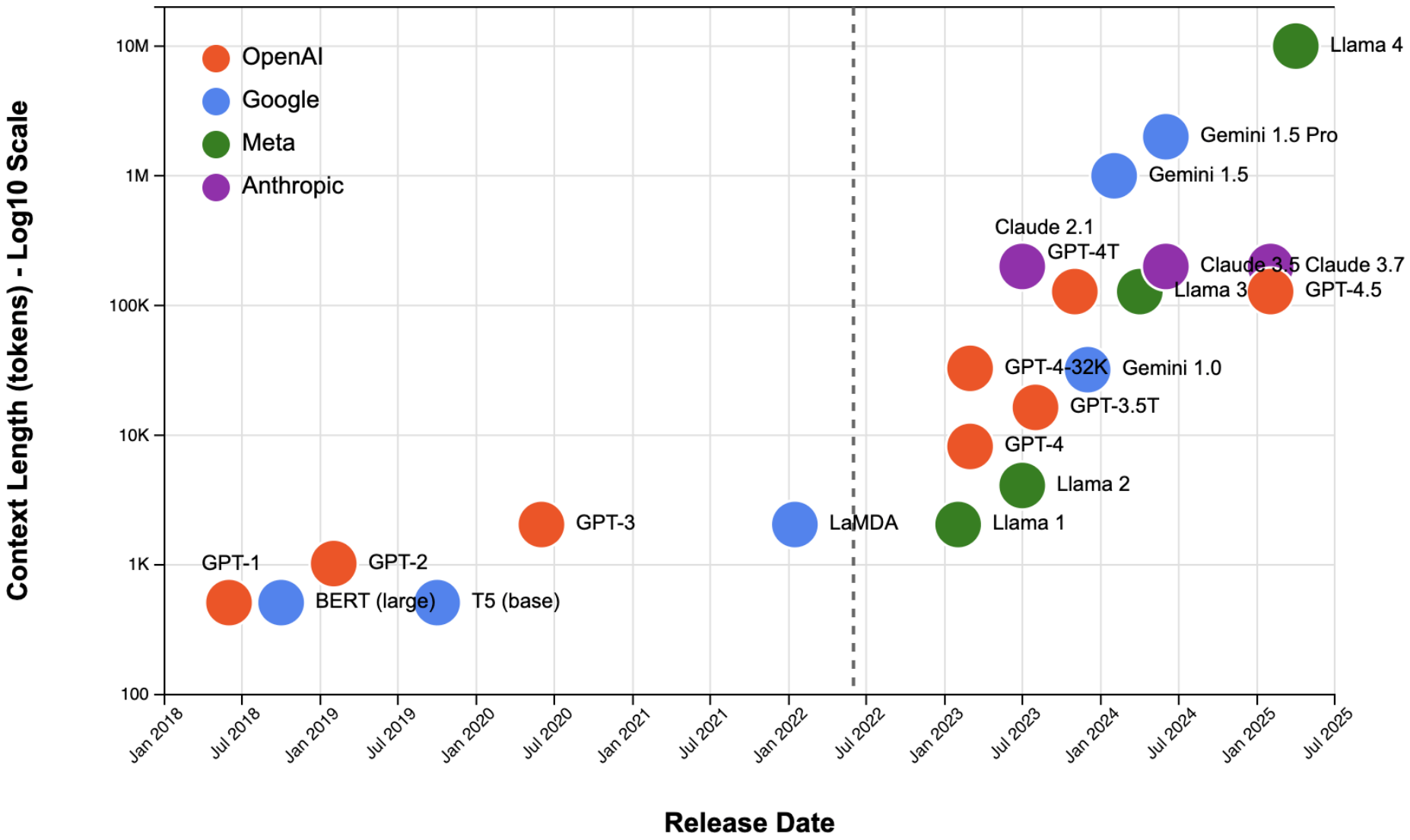
	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
Model	1K	2K	4K	8K	16K	1K	2K	4K	8K
Softmax	**100.0**	99.4	94.2	11.4	0.8	**100.0**	**100.0**	4.8	0.0
SSMax	**100.0**	99.8	99.2	92.0	75.2	99.4	99.2	64.4	14.8
Entmax	99.8	99.8	89.0	21.6	1.2	99.6	99.4	64.8	7.2
ASEntmax	99.6	**100.0**	**100.0**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

RULER – Needle in a haystack

	S-NIAH-1 (ID)		S-NIAH-1 (OOD)			S-NIAH-2 (ID)		S-NIAH-2 (OOD)	
Model	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**

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**

Context Length of Current LLMs



But...

Model	Parameters	Jump Factor	Chinchilla Tokens (B)	Jump Factor	CS-2 Config	Days To Train	Jump Factor	Price To Train	Jump Factor	Cost Per 1M Parameters
GPT-3XL	1.3		26		4 * CS-2	0.4		$2,500		$1.92
GPT-J	6	4.6 X	120	4.6 X	4 * CS-2	8	20.0 X	$45,000	18.0 X	$7.50
GPT-3 6.7B	6.7	1.1 X	134	1.1 X	4 * CS-2	11	1.4 X	$40,000	0.9 X	$5.97
T-5 11B	11	1.6 X	*34*	0.3 X	4 * CS-2	9	0.8 X	$60,000	1.5 X	$5.45
GPT-3 13B	13	1.2 X	260	7.6 X	4 * CS-2	39	4.3 X	$150,000	2.5 X	$11.54
GPT NeoX	20	1.5 X	400	1.5 X	4 * CS-2	47	1.2 X	$525,000	3.5 X	$26.25
GPT NeoX	*20*	*1.5 X*	***400***	*1.5 X*	***16 * CS-2***	***11.1***	*0.3 X*	***$656,250***	*4.4 X*	***$32.81***
GPT 70B	70	3.5 X	1,400	3.5 X	4 * CS-2	85	1.8 X	$2,500,000	4.8 X	$35.71
GPT 70B	*70*	*3.5 X*	***1,400***	*3.5 X*	***16 * CS-2***	***21.3***	*0.3 X*	***$3,125,000***	*6.0 X*	***$44.64***
GPT 175B	175	2.5 X	3,500	2.5 X	4 * CS-2	***110.5***	1.3 X	***$8,750,000***	3.5 X	***$50.00***
GPT 175B	*175*	*2.5 X*	***3,500***	*2.5 X*	***16 * CS-2***	***27.6***	*0.3 X*	***$10,937,500***	*4.4 X*	***$62.50***

[source]

But...

Model	Parameters	Jump Factor	Chinchilla Tokens (B)	Jump Factor	CS-2 Config	Days To Train	Jump Factor	Price To Train	Jump Factor	Cost Per 1M Parameters
GPT-3XL	1.3		26		4 * CS-2	0.4		$2,500		$1.92
GPT-J	6	4.6 X	120	4.6 X	4 * CS-2	8	20.0 X	$45,000	18.0 X	$7.50
GPT-3 6.7B	6.7	1.1 X	134	1.1 X	4 * CS-2	11	1.4 X	$40,000	0.9 X	$5.97
T-5 11B	11	1.6 X	*34*	0.3 X	4 * CS-2	9	0.8 X	$60,000	1.5 X	$5.45
GPT-3 13B	13	1.2 X	260	7.6 X	4 * CS-2	39	4.3 X	$150,000	2.5 X	$11.54
GPT NeoX	20	1.5 X	400	1.5 X	4 * CS-2	47	1.2 X	$525,000	3.5 X	$26.25
GPT NeoX	*20*	*1.5 X*	***400***	*1.5 X*	***16 * CS-2***	***11.1***	*0.3 X*	***$656,250***	*4.4 X*	***$32.81***
GPT 70B	70	3.5 X	1,400	3.5 X	4 * CS-2	85	1.8 X	$2,500,000	4.8 X	$35.71
GPT 70B	*70*	*3.5 X*	***1,400***	*3.5 X*	***16 * CS-2***	***21.3***	*0.3 X*	***$3,125,000***	*6.0 X*	***$44.64***
GPT 175B	175	2.5 X	3,500	2.5 X	4 * CS-2	***110.5***	1.3 X	***$8,750,000***	3.5 X	***$50.00***
GPT 175B	*175*	*2.5 X*	***3,500***	*2.5 X*	***16 * CS-2***	***27.6***	*0.3 X*	***$10,937,500***	*4.4 X*	***$62.50***

[source]

But...

Model	Parameters	Jump Factor	Chinchilla Tokens (B)	Jump Factor	CS-2 Config	Days To Train	Jump Factor	Price To Train	Jump Factor	Cost Per 1M Parameters
GPT-3XL	1.3		26		4 * CS-2	0.4		$2,500		$1.92
GPT-J	6	4.6 X	120	4.6 X	4 * CS-2	8	20.0 X	$45,000	18.0 X	$7.50
GPT-3 6.7B	6.7	1.1 X	134	1.1 X	4 * CS-2	11	1.4 X	$40,000	0.9 X	$5.97
T-5 11B	11	1.6 X	*34*	0.3 X	4 * CS-2	9	0.8 X	$60,000	1.5 X	$5.45
GPT-3 13B	13	1.2 X	260	7.6 X	4 * CS-2	39	4.3 X	$150,000	2.5 X	$11.54
GPT NeoX	20	1.5 X	400	1.5 X	4 * CS-2	47	1.2 X	$525,000	3.5 X	$26.25
GPT NeoX	*20*	*1.5 X*	***400***	*1.5 X*	***16 * CS-2***	***11.1***	*0.3 X*	***$656,250***	*4.4 X*	***$32.81***
GPT 70B	70	3.5 X	1,400	3.5 X	4 * CS-2	85	1.8 X	$2,500,000	4.8 X	$35.71
GPT 70B	*70*	*3.5 X*	***1,400***	*3.5 X*	***16 * CS-2***	***21.3***	*0.3 X*	***$3,125,000***	*6.0 X*	***$44.64***
GPT 175B	175	2.5 X	3,500	2.5 X	4 * CS-2	***110.5***	1.3 X	***$8,750,000***	3.5 X	***$50.00***
GPT 175B	*175*	*2.5 X*	***3,500***	*2.5 X*	***16 * CS-2***	***27.6***	*0.3 X*	***$10,937,500***	*4.4 X*	***$62.50***

[source]

But...

Model	Parameters	Jump Factor	Chinchilla Tokens (B)	Jump Factor	CS-2 Config	Days To Train	Jump Factor	Price To Train	Jump Factor	Cost Per 1M Parameters
GPT-3XL	1.3		26		4 * CS-2	0.4		$2,500		$1.92
GPT-J	6	4.6 X	120	4.6 X	4 * CS-2	8	20.0 X	$45,000	18.0 X	$7.50
GPT-3 6.7B	6.7	1.1 X	134	1.1 X	4 * CS-2	11	1.4 X	$40,000	0.9 X	$5.97
T-5 11B	11	1.6 X	*34*	0.3 X	4 * CS-2	9	0.8 X	$60,000	1.5 X	$5.45
GPT-3 13B	13	1.2 X	260	7.6 X	4 * CS-2	39	4.3 X	$150,000	2.5 X	$11.54
GPT NeoX	20	1.5 X	400	1.5 X	4 * CS-2	47	1.2 X	$525,000	3.5 X	$26.25
GPT NeoX	*20*	*1.5 X*	***400***	*1.5 X*	***16 * CS-2***	***11.1***	*0.3 X*	***$656,250***	*4.4 X*	***$32.81***
GPT 70B	70	3.5 X	1,400	3.5 X	4 * CS-2	85	1.8 X	$2,500,000	4.8 X	$35.71
GPT 70B	*70*	*3.5 X*	***1,400***	*3.5 X*	***16 * CS-2***	***21.3***	*0.3 X*	***$3,125,000***	*6.0 X*	***$44.64***
GPT 175B	175	2.5 X	3,500	2.5 X	4 * CS-2	***110.5***	1.3 X	***$8,750,000***	3.5 X	***$50.00***
GPT 175B	*175*	*2.5 X*	***3,500***	*2.5 X*	***16 * CS-2***	***27.6***	*0.3 X*	***$10,937,500***	*4.4 X*	***$62.50***

[source]

Model name	Number of parameters	Datacenter PUE	Carbon intensity of grid used	Power consumption	CO_2eq emissions	CO_2eq emissions × PUE
GPT-3	175B	1.1	429 gCO_2eq/kWh	1,287 MWh	*502 tonnes*	552 tonnes
Gopher	280B	1.08	330 gCO_2eq/kWh	*1,066 MWh*	*352 tonnes*	380 tonnes
OPT	175B	*1.09* [2]	*231 gCO_2eq/kWh*	*324 MWh*	70 tonnes	*76.3 tonnes* [3]
BLOOM	176B	1.2	57 gCO_2eq/kWh	433 MWh	25 tonnes	30 tonnes

[source]

~112 homes per year /
~3M km / (8 L/km)

But...

Model	Parameters	Jump Factor	Chinchilla Tokens (B)	Jump Factor	CS-2 Config	Days To Train	Jump Factor	Price To Train	Jump Factor	Cost Per 1M Parameters
GPT-3XL	1.3		26		4 * CS-2	0.4		$2,500		$1.92
GPT-J	6	4.6 X	120	4.6 X	4 * CS-2	8	20.0 X	$45,000	18.0 X	$7.50
GPT-3 6.7B	6.7	1.1 X	134	1.1 X	4 * CS-2	11	1.4 X	$40,000	0.9 X	$5.97
T-5 11B	11	1.6 X	*34*	0.3 X	4 * CS-2	9	0.8 X	$60,000	1.5 X	$5.45
GPT-3 13B	13	1.2 X	260	7.6 X	4 * CS-2	39	4.3 X	$150,000	2.5 X	$11.54
GPT NeoX	20	1.5 X	400	1.5 X	4 * CS-2	47	1.2 X	$525,000	3.5 X	$26.25
GPT NeoX	*20*	*1.5 X*	***400***	*1.5 X*	***16 * CS-2***	***11.1***	*0.3 X*	***$656,250***	*4.4 X*	***$32.81***
GPT 70B	70	3.5 X	1,400	3.5 X	4 * CS-2	85	1.8 X	$2,500,000	4.8 X	$35.71
GPT 70B	*70*	*3.5 X*	***1,400***	*3.5 X*	***16 * CS-2***	***21.3***	*0.3 X*	***$3,125,000***	*6.0 X*	***$44.64***
GPT 175B	175	2.5 X	3,500	2.5 X	4 * CS-2	***110.5***	1.3 X	***$8,750,000***	3.5 X	***$50.00***
GPT 175B	*175*	*2.5 X*	***3,500***	*2.5 X*	***16 * CS-2***	***27.6***	*0.3 X*	***$10,937,500***	*4.4 X*	***$62.50***

[source]

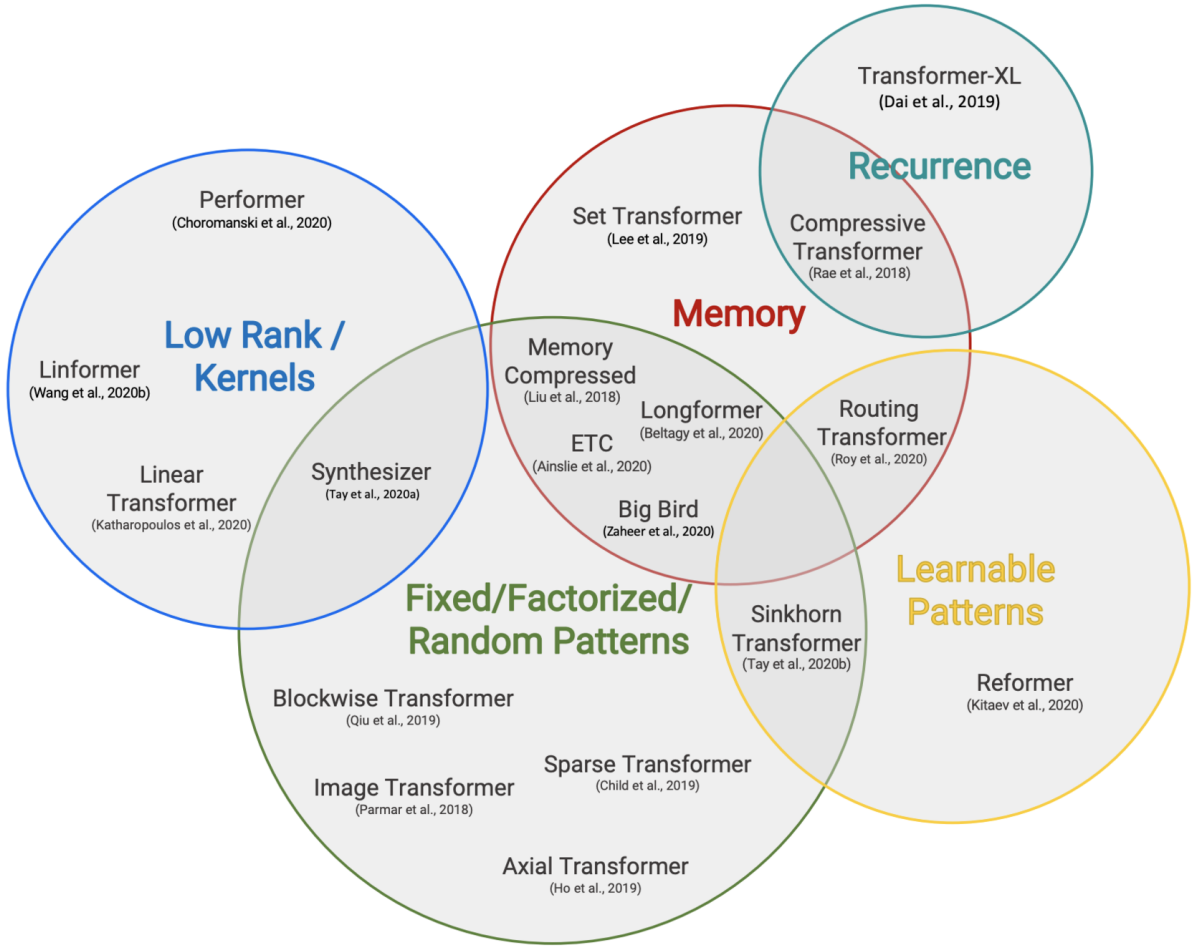
Model name	Number of parameters	Datacenter PUE	Carbon intensity of grid used	Power consumption	CO_2eq emissions	CO_2eq emissions × PUE
GPT-3	175B	1.1	429 gCO_2eq/kWh	1,287 MWh	*502 tonnes*	552 tonnes
Gopher	280B	1.08	330 gCO_2eq/kWh	*1,066 MWh*	*352 tonnes*	380 tonnes
OPT	175B	*1.09* [2]	*231 gCO_2eq/kWh*	*324 MWh*	70 tonnes	*76.3 tonnes* [3]
BLOOM	176B	1.2	57 gCO_2eq/kWh	433 MWh	25 tonnes	30 tonnes

[source]

~112 homes per year /
~3M km / (8 L/km)

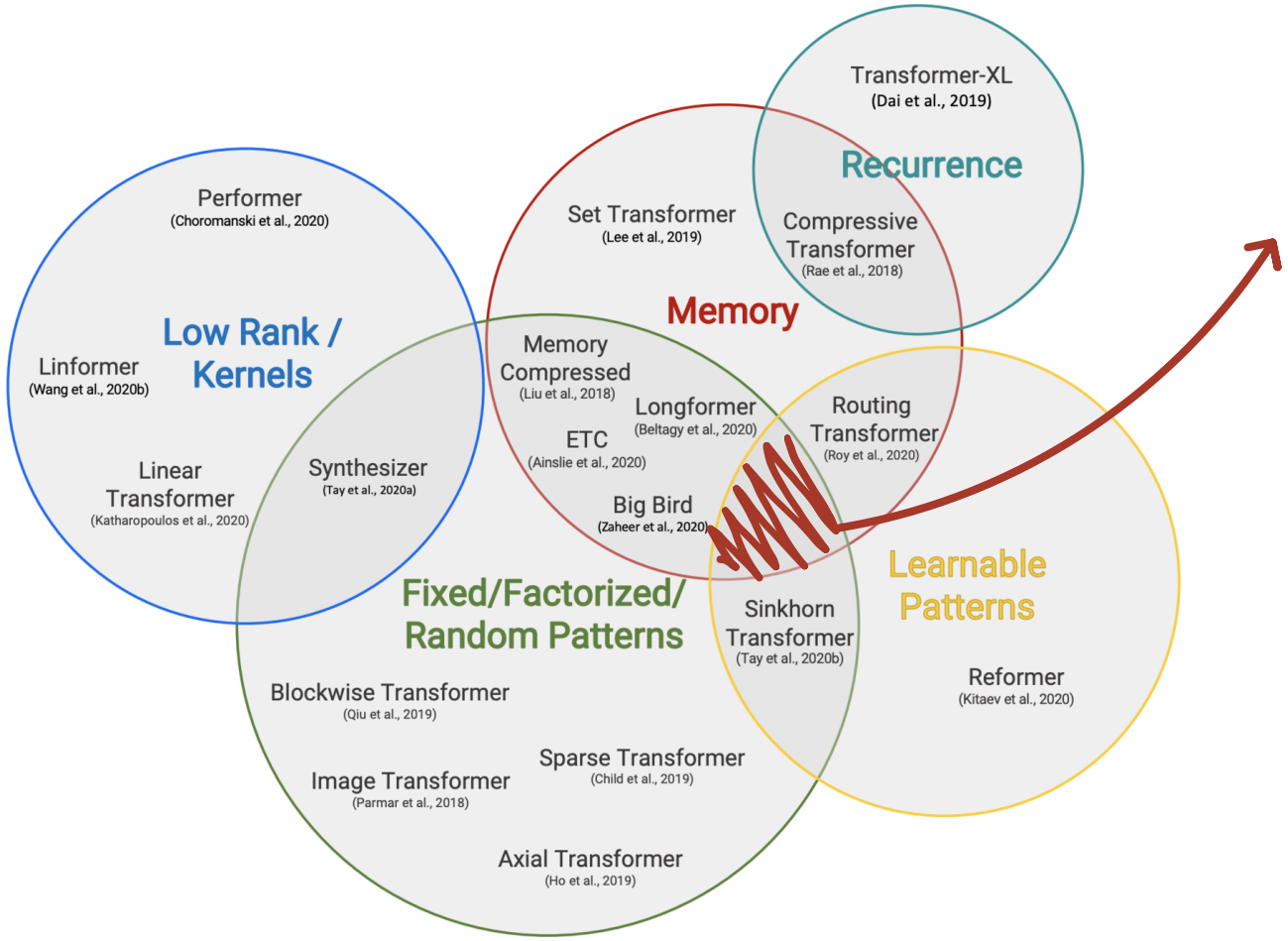
🚨 Just **a single** training run!

Efficient Attention to the Rescue



(Tay et al., 2020)

Efficient Attention to the Rescue



Dream spot:

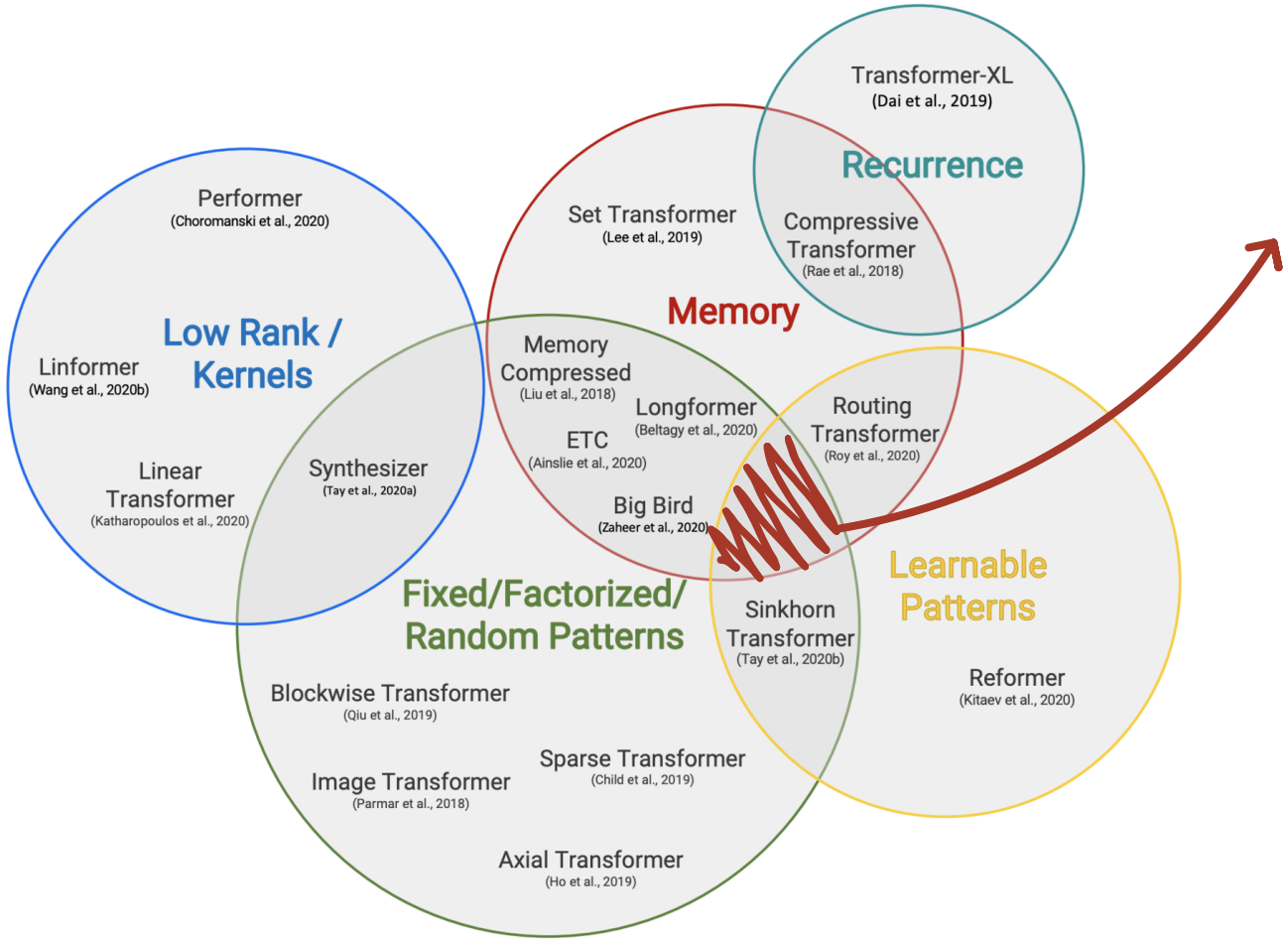
Sparse

Adaptive

Flexible

(Tay et al., 2020)

Efficient Attention to the Rescue



Dream spot:

Sparse

Adaptive

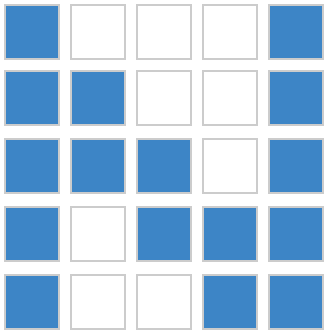
Flexible

🚨 Not used in practice! Why? They trade-off accuracy for efficiency!

(Tay et al., 2020)

The Types of Sparse Attention

Fixed Patterns



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

😕 **Low Adaptivity**

😕 **Low Flexibility**

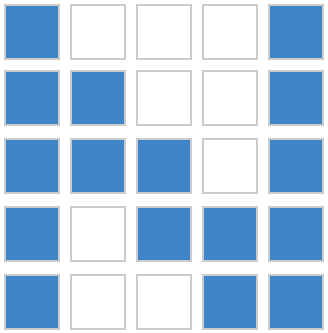
😊 **Full Differentiability**

🚀 $O(n \times k)$ time cost

$O(n \times k)$ memory cost

The Types of Sparse Attention

Fixed Patterns



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

😕 **Low Adaptivity**

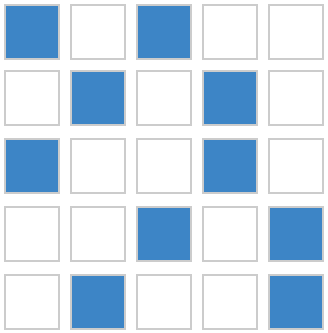
😕 **Low Flexibility**

😊 **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**

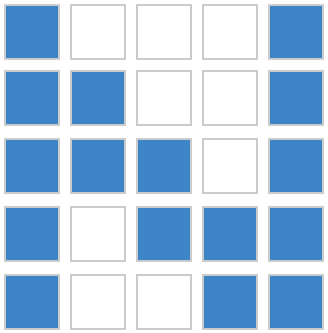
😕 **Approx or No Differentiability**

🛩️ $O(n \log n)$ time cost

$O(n \log n)$ memory cost

The Types of Sparse Attention

Fixed Patterns



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

😕 **Low Adaptivity**

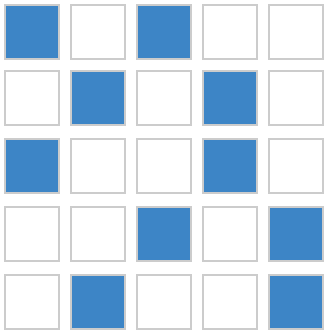
😕 **Low Flexibility**

😊 **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**

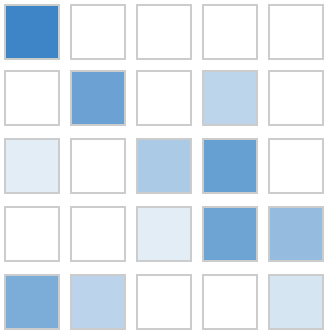
😐 **Medium Flexibility**

😕 **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 sparisty

😊 **High Adaptivity**

😊 **High Flexibility**

😊 **Full Differentiability**

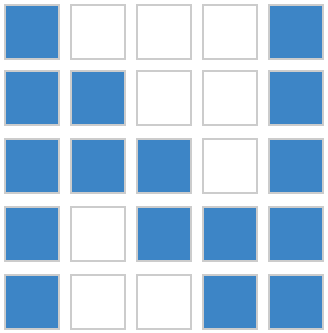
🛵 $O(n^2 \log n)$ time cost

$O(n^2)$ memory cost

but $O(n)$ in theory

The Types of Sparse Attention

Fixed Patterns



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

😕 **Low Adaptivity**

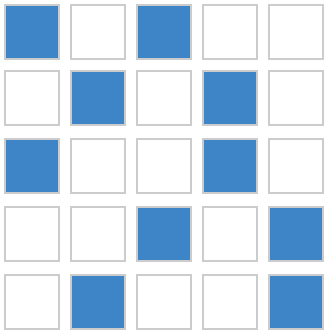
😕 **Low Flexibility**

😊 **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**

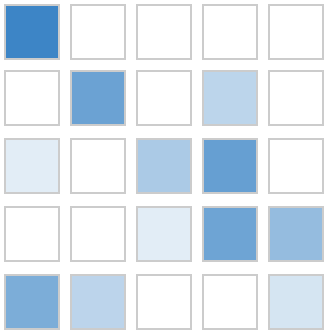
😐 **Medium Flexibility**

😕 **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 sparisty

😊 **High Adaptivity**

😊 **High Flexibility**

😊 **Full Differentiability**

🛵 $O(n^2 \log n)$ time cost

$O(n^2)$ memory cost

but $O(n)$ in theory

Computing α-entmax

$$\boldsymbol{p} = \max(0, (\alpha - 1)\boldsymbol{z} - \tau)^{\frac{1}{\alpha-1}} \quad \text{such that } \sum_i p_i = 1$$

- $\alpha = 1.5$

Sort $\boldsymbol{z}$, yielding $z_{[d]} \leq \cdots \leq z_{[1]}$; set $\boldsymbol{z} \leftarrow \boldsymbol{z}/2$
for $\rho \in 1, \ldots, d$ **do**
  $M(\rho) \leftarrow 1/\rho \sum_{j=1}^{\rho} z_{[j]}$
  $S(\rho) \leftarrow \sum_{j=1}^{\rho} \left(z_{[j]} - M(\rho)\right)^2$
  $\tau(\rho) \leftarrow M(\rho) - \sqrt{1/\rho\,(1 - S(\rho))}$
  if $z_{[\rho+1]} \leq \tau(\rho) \leq z_{[\rho]}$ **then**
    return $\boldsymbol{p}^\star = [\boldsymbol{z} - \tau\,\mathbf{1}]_+^2$

(Peters, Niculae, and Martins, 2019)

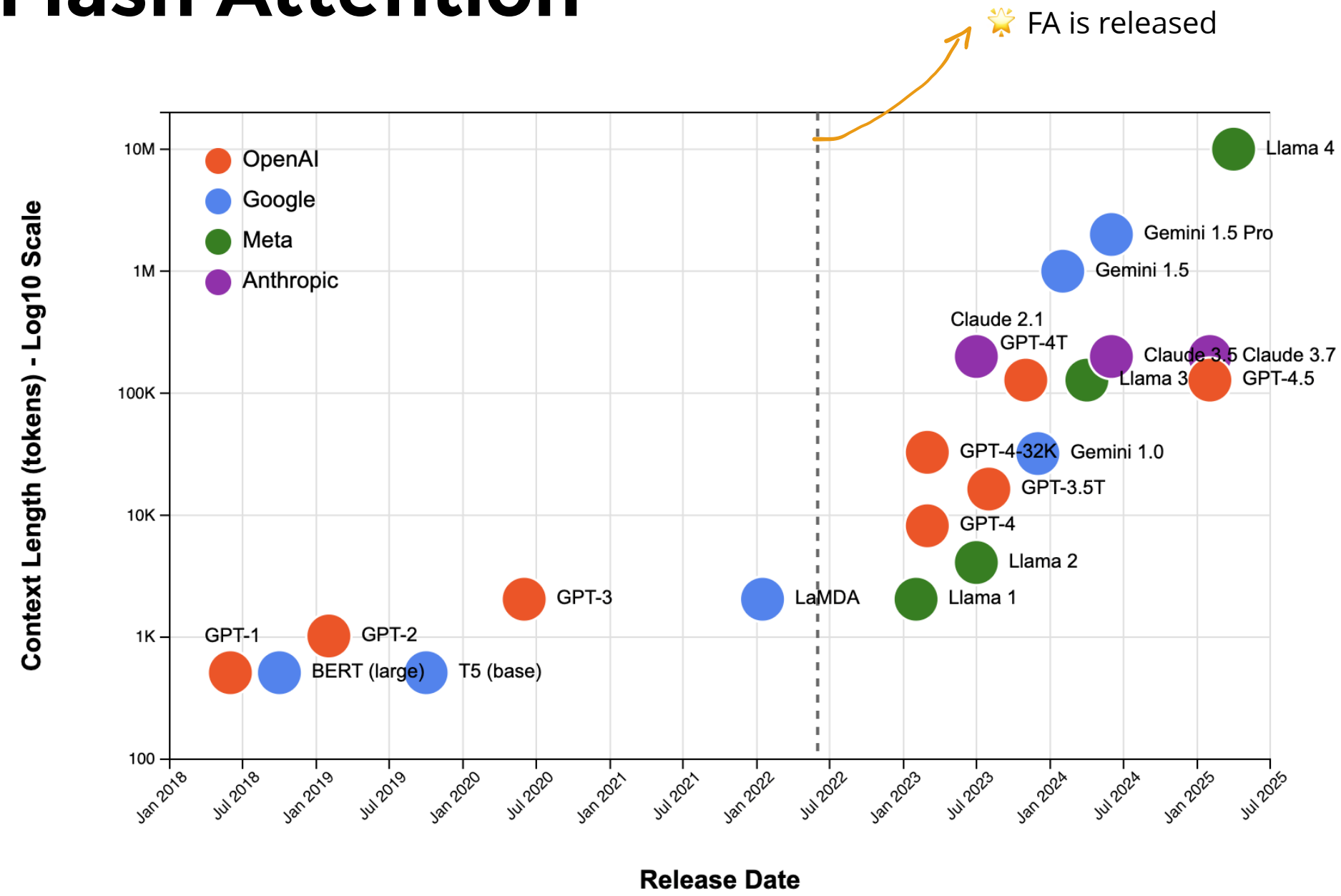
- $\alpha = 2$ (sparsemax)

Input: $\boldsymbol{z}$
Sort $\boldsymbol{z}$ as $z_{(1)} \geq \ldots \geq z_{(K)}$
Find $k(\boldsymbol{z}) := \max \left\{ k \in [K] \mid 1 + kz_{(k)} > \sum_{j \leq k} z_{(j)} \right\}$
Define $\tau(\boldsymbol{z}) = \frac{\left(\sum_{j \leq k(\boldsymbol{z})} z_{(j)}\right) - 1}{k(\boldsymbol{z})}$
Output: $\boldsymbol{p}$ s.t. $p_i = [z_i - \tau(\boldsymbol{z})]_+$.

(Martins and Astudillo, 2016)

- Does not work for all values of α
- Requires sorting (not GPU-friendly)

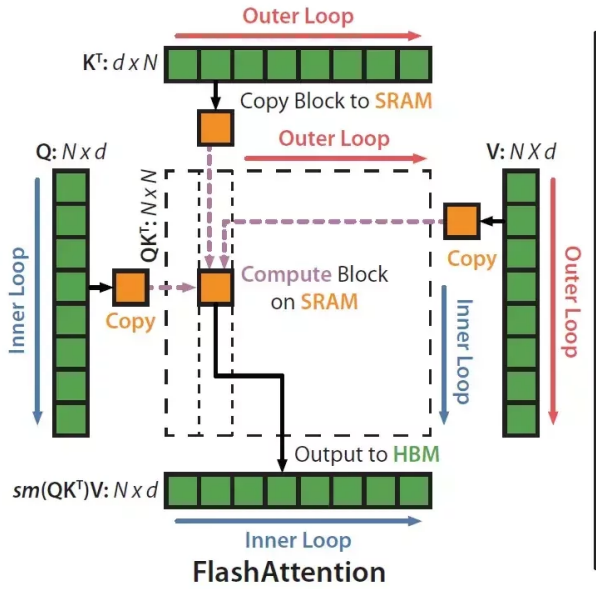
⚡Flash Attention

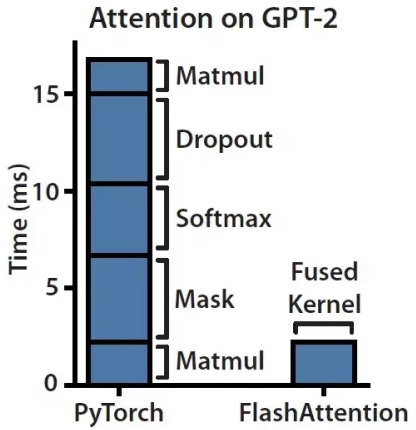


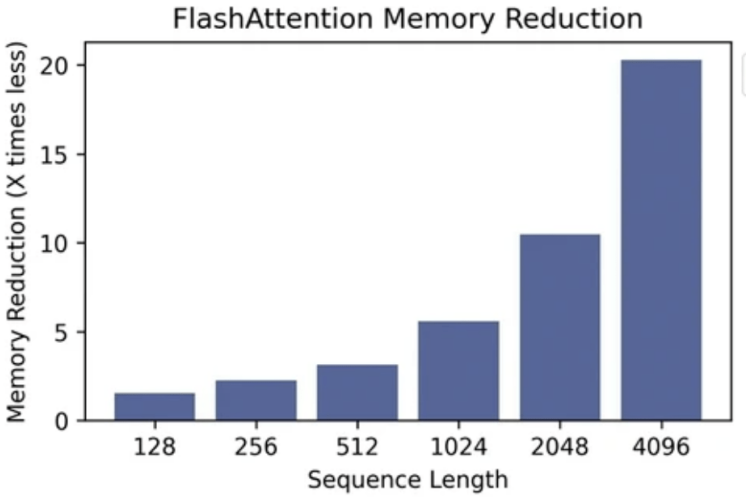
⚡Flash Attention

- **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







⚡AdaSplash: Adaptive Sparse Flash Attention


Nuno Gonçalves


Marcos Treviso


André F. T. Martins

📄 arxiv.org/abs/2502.12082

💻 github.com/deep-spin/adasplash

```python
# pip install adasplash
from adasplash import adasplash

y = adasplash(q, k, v, alpha=1.5, niter=3, is_causal=True)
```

Computing α-entmax

$$\alpha\text{-entmax}(z) = [(\alpha - 1)z_i - \tau]_+^{\frac{1}{\alpha-1}}$$

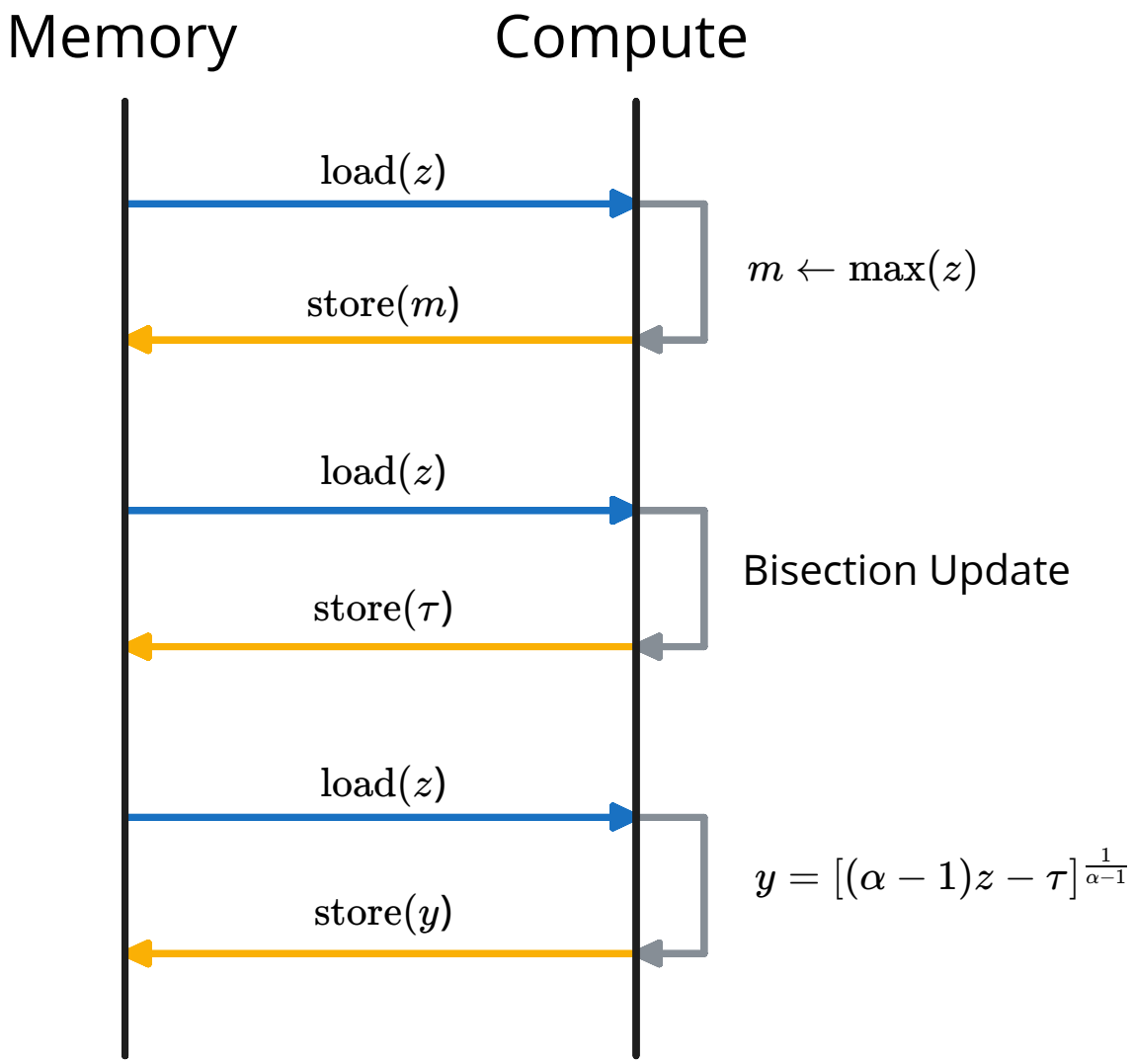
We need to find τ in such a way that our vector sums to 1.

$$f(\tau) = \sum_i [(\alpha - 1)z_i - \tau]_+^{\frac{1}{\alpha-1}} - 1$$

This is a just a **root-finding** problem!

Our τ will be the root of $f(\tau)$

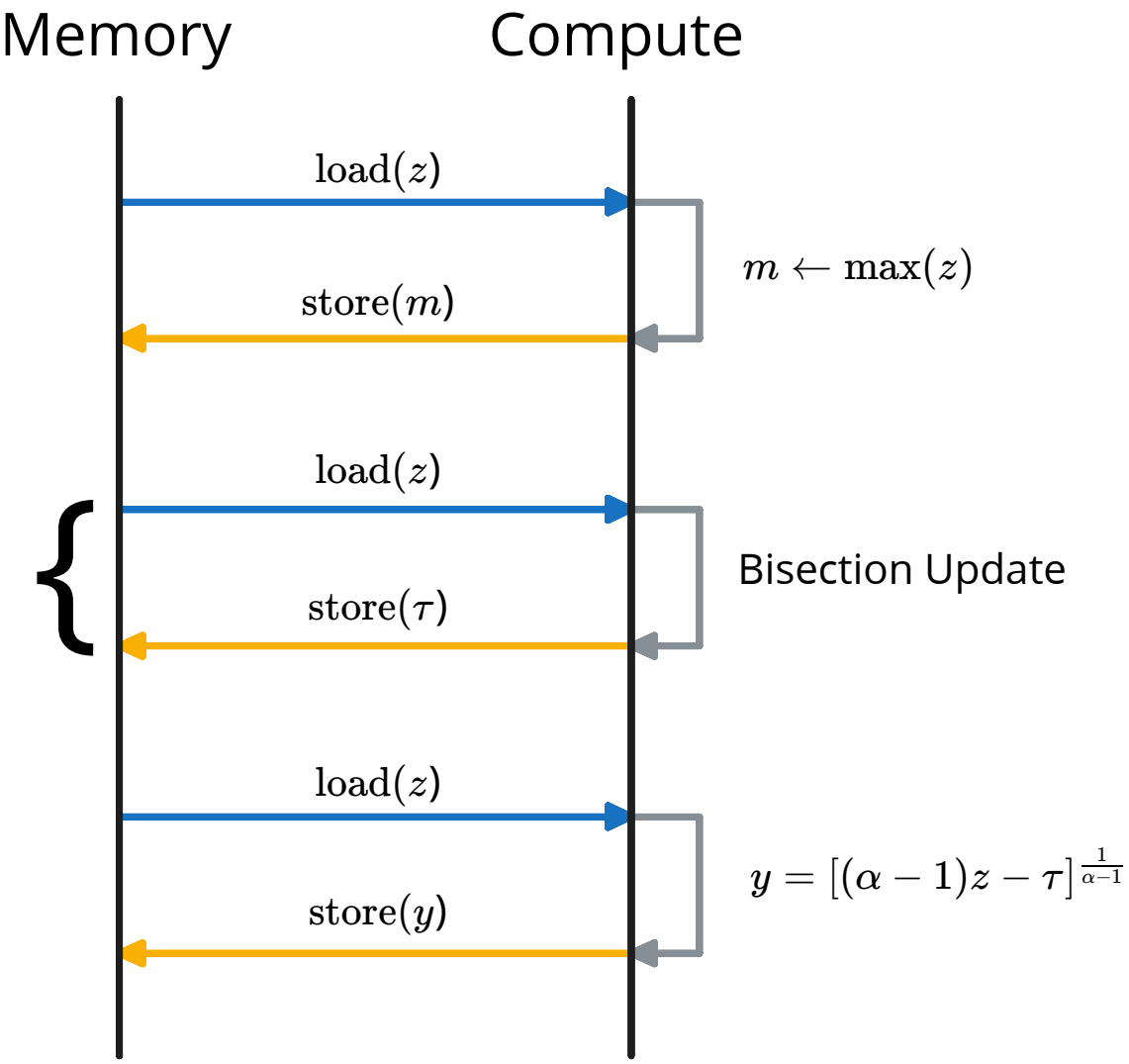
Computing α-entmax



Computing α-entmax

Main Bottleneck!

Requires ~20 iterations ☹



Can we do better than Bisection?

$$f^{(m)}(\tau) = \underbrace{(-1)^m \frac{\Gamma\left(\frac{1}{\alpha-1}+1\right)}{\Gamma\left(\frac{1}{\alpha-1}-m+1\right)}}_{\text{constant}} \sum_i \Big[(\alpha-1)z_i - \tau\Big]_+^{\frac{1}{\alpha-1}-m}$$

"shifted" power

Can we do better than Bisection?

$$f^{(m)}(\tau) = \underbrace{(-1)^m \frac{\Gamma\left(\frac{1}{\alpha-1}+1\right)}{\Gamma\left(\frac{1}{\alpha-1}-m+1\right)}}_{\text{constant}} \sum_i \Big[(\alpha-1)z_i - \tau\Big]_+^{\frac{1}{\alpha-1}-m}$$

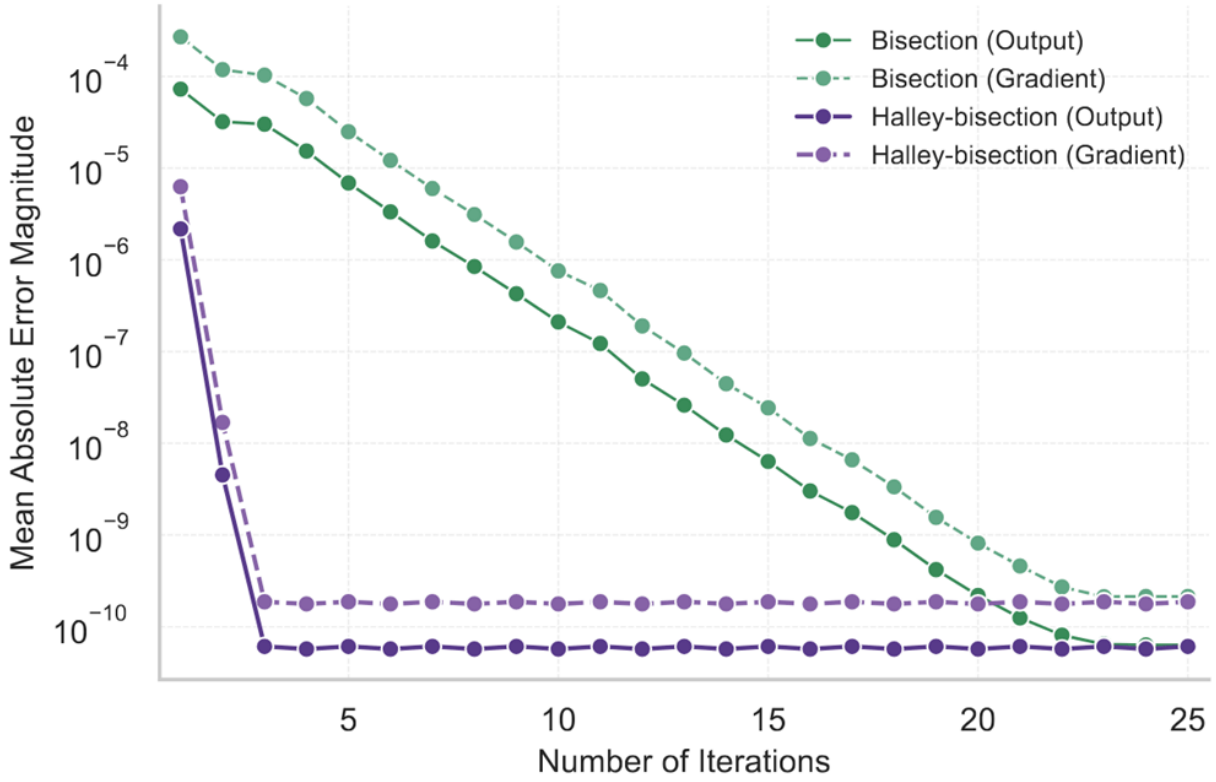
"shifted" power

Halley's Method

$$H_f(\tau) = \tau - \frac{2f(\tau)\,f'(\tau)}{2(f'(\tau))^2 - f(\tau)\,f''(\tau)}$$

$$f'(\tau) = -\frac{1}{\alpha-1}\sum_i \big[(\alpha-1)z_i - \tau\big]_+^{1/(\alpha-1)-1} \qquad f''(\tau) = \frac{2-\alpha}{\alpha-1}\sum_i \big[(\alpha-1)z_i - \tau\big]_+^{1/(\alpha-1)-2}$$

Proposed Solution



7x reduction on the number of iterations

⚡α-entmax Flash Attention

Algorithm 1 α-entmax Attention Kernel

Require: Fast shared memory of size M, sparsity control $\alpha \in (1, \infty]$.
Input: $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$, keys $\boldsymbol{K} \in \mathbb{R}^{N \times d}$, and values $\boldsymbol{V} \in \mathbb{R}^{N \times d}$.

Initialize $\boldsymbol{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \boldsymbol{\tau} = (0)_N \in \mathbb{R}^N$ in slow memory.
Divide $\boldsymbol{Q}, \boldsymbol{\tau}, \boldsymbol{O}$ into T_r blocks of size $\lceil \frac{M}{4} \rceil$
Divide $\boldsymbol{K}, \boldsymbol{V}$ into T_c blocks of size $\lceil \frac{M}{4} \rceil$
for $1 \leq i \leq T_r$ **do**
    Load $\boldsymbol{Q}_i, \boldsymbol{O}_i, \boldsymbol{\tau}_i$ from slow to fast memory
    Compute $\boldsymbol{\tau}_i(\alpha)$ on fast memory via Halley-bisection
    for $1 \leq j \leq T_c$ **do**
        Load $\boldsymbol{K}_j, \boldsymbol{V}_j$ from slow to fast memory
        Compute $\boldsymbol{Z}_{ij} = \boldsymbol{Q}_i \boldsymbol{K}_j^T$
        Compute $\boldsymbol{P}_{ij} = \max(0, (\alpha - 1)\boldsymbol{Z}_{ij} - \boldsymbol{\tau}_i)^{1/(\alpha-1)}$
        Write $\boldsymbol{O}_i = \boldsymbol{O}_i + \boldsymbol{P}_{ij} \boldsymbol{V}_j$ to slow memory
    end for
end for
Return $\boldsymbol{O}$.

⚡α-entmax Flash Attention

Algorithm 1 α-entmax Attention Kernel

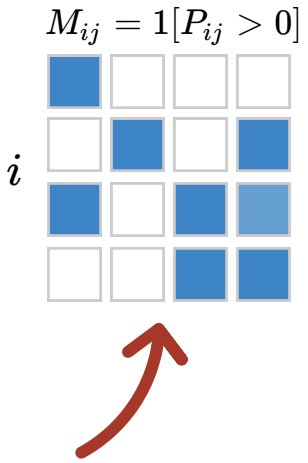
Require: Fast shared memory of size M, sparsity control $\alpha \in (1, \infty]$.
Input: $\boldsymbol{Q} \in \mathbb{R}^{N \times d}$, keys $\boldsymbol{K} \in \mathbb{R}^{N \times d}$, and values $\boldsymbol{V} \in \mathbb{R}^{N \times d}$.

Initialize $\boldsymbol{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \boldsymbol{\tau} = (0)_N \in \mathbb{R}^N$ in slow memory.
Divide $\boldsymbol{Q}, \boldsymbol{\tau}, \boldsymbol{O}$ into T_r blocks of size $\lceil \frac{M}{4} \rceil$
Divide $\boldsymbol{K}, \boldsymbol{V}$ into T_c blocks of size $\lceil \frac{M}{4} \rceil$
for $1 \leq i \leq T_r$ **do**
    Load $\boldsymbol{Q}_i, \boldsymbol{O}_i, \boldsymbol{\tau}_i$ from slow to fast memory
    Compute $\boldsymbol{\tau}_i(\alpha)$ on fast memory via Halley-bisection
    for $1 \leq j \leq T_c$ **do**
        Load $\boldsymbol{K}_j, \boldsymbol{V}_j$ from slow to fast memory
        Compute $\boldsymbol{Z}_{ij} = \boldsymbol{Q}_i \boldsymbol{K}_j^T$
        Compute $\boldsymbol{P}_{ij} = \max(0, (\alpha - 1)\boldsymbol{Z}_{ij} - \boldsymbol{\tau}_i)^{1/(\alpha-1)}$
        Write $\boldsymbol{O}_i = \boldsymbol{O}_i + \boldsymbol{P}_{ij} \boldsymbol{V}_j$ to slow memory
    end for
end for
Return $\boldsymbol{O}$.

🚨 How to leverage sparsity to speed up computation?

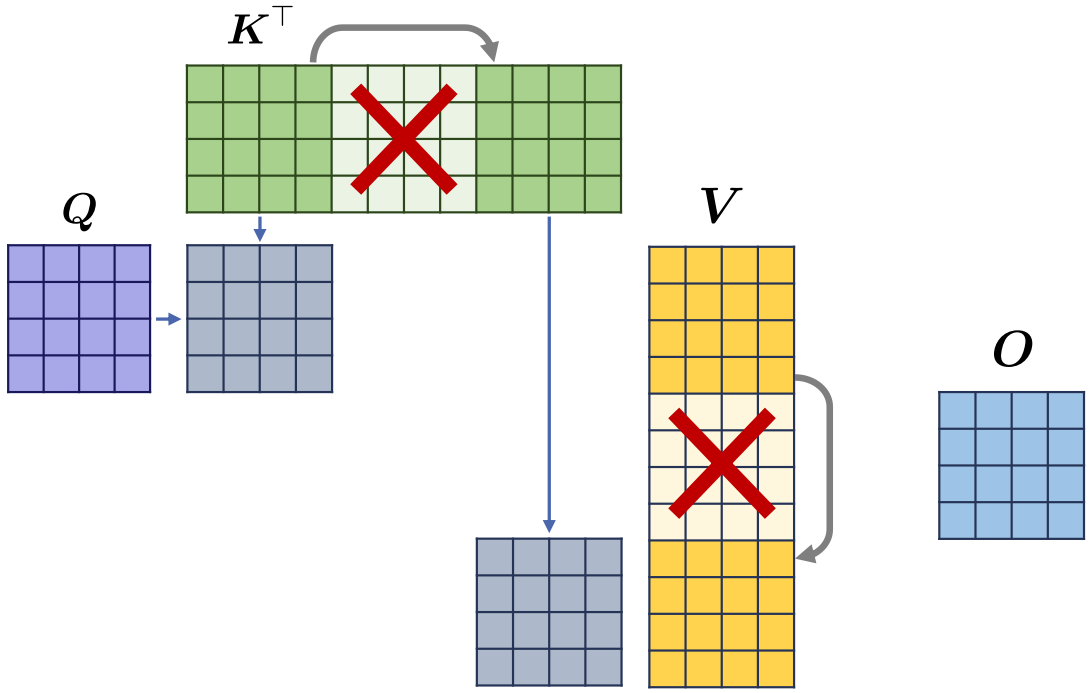
⚡AdaSplash: Adaptive Sparse Flash Attention

The goal is to keep refining τ and and use the sparsity to skip redundant loads + computation.

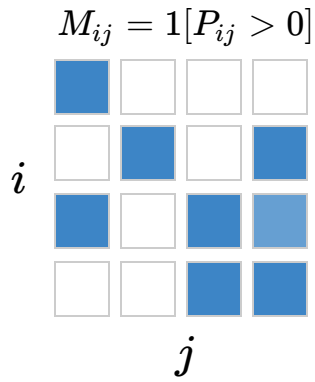


"Dynamic" block mask

Keep track of which block have non-zeros.



⚡AdaSplash: Adaptive Sparse Flash Attention



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

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

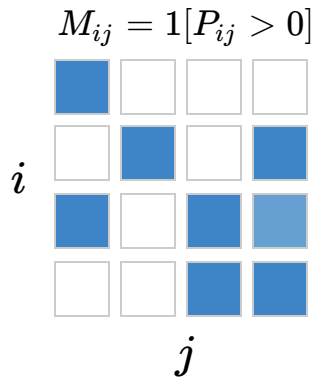
⚡ Efficient backward pass!

$$\boldsymbol{dQ}_i = \sum_{j \in \mathcal{Q}_i} \boldsymbol{dS}_i^{(j)} \cdot \boldsymbol{K}_j$$

$$\boldsymbol{dK}_j = \sum_{i \in \mathcal{K}_j} \boldsymbol{dS}_i^{(j)} \cdot \boldsymbol{Q}_i$$

$$\boldsymbol{dV}_j = \sum_{i \in \mathcal{K}_j} \boldsymbol{P}_i^{(j)} \cdot \boldsymbol{dO}_i$$

⚡AdaSplash: Adaptive Sparse Flash Attention



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

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

⚡ Efficient backward pass!

$$\boldsymbol{dQ}_i = \sum_{j \in \mathcal{Q}_i} \boldsymbol{dS}_i^{(j)} \cdot \boldsymbol{K}_j$$

$$\boldsymbol{dK}_j = \sum_{i \in \mathcal{K}_j} \boldsymbol{dS}_i^{(j)} \cdot \boldsymbol{Q}_i$$

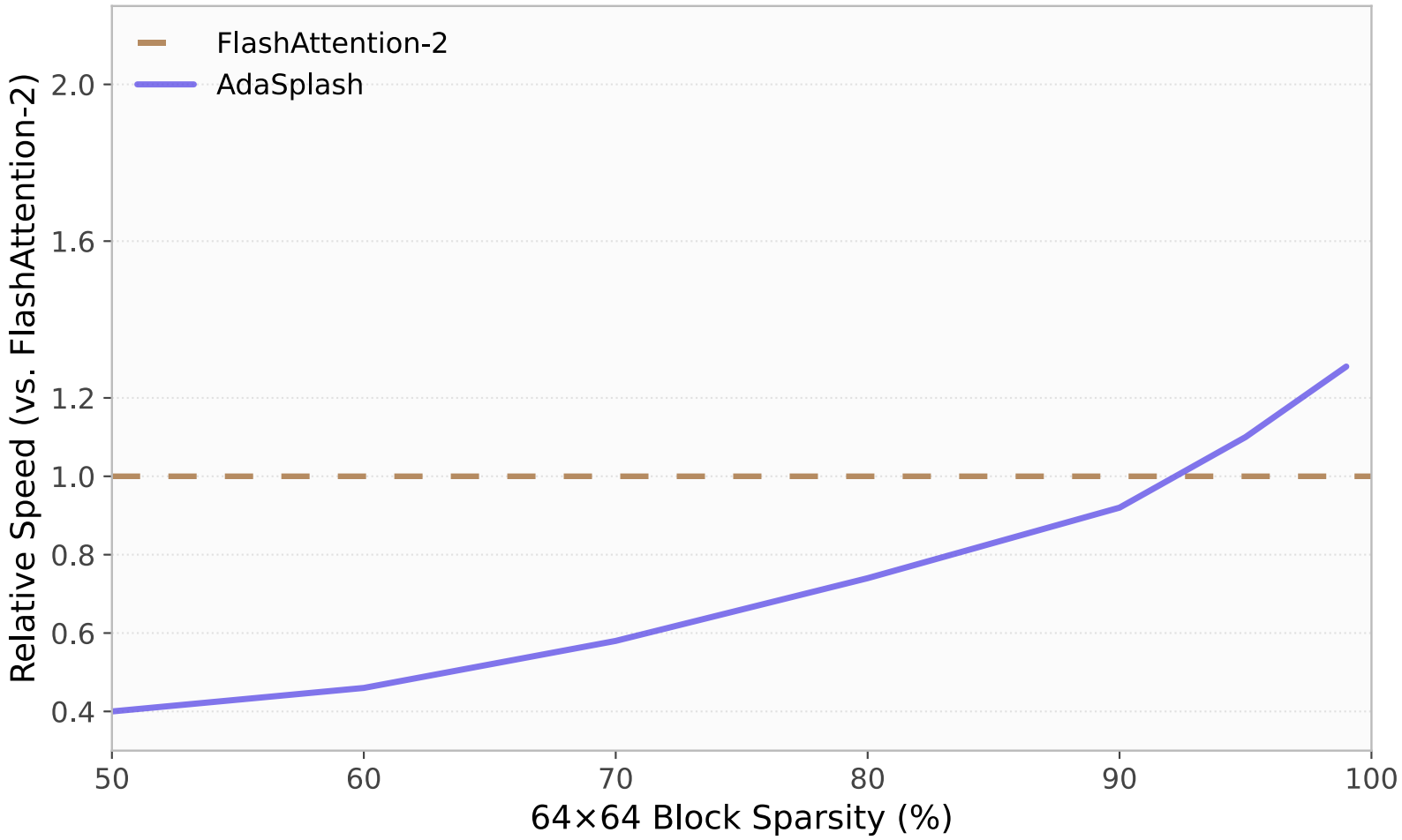
$$\boldsymbol{dV}_j = \sum_{i \in \mathcal{K}_j} \boldsymbol{P}_i^{(j)} \cdot \boldsymbol{dO}_i$$

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

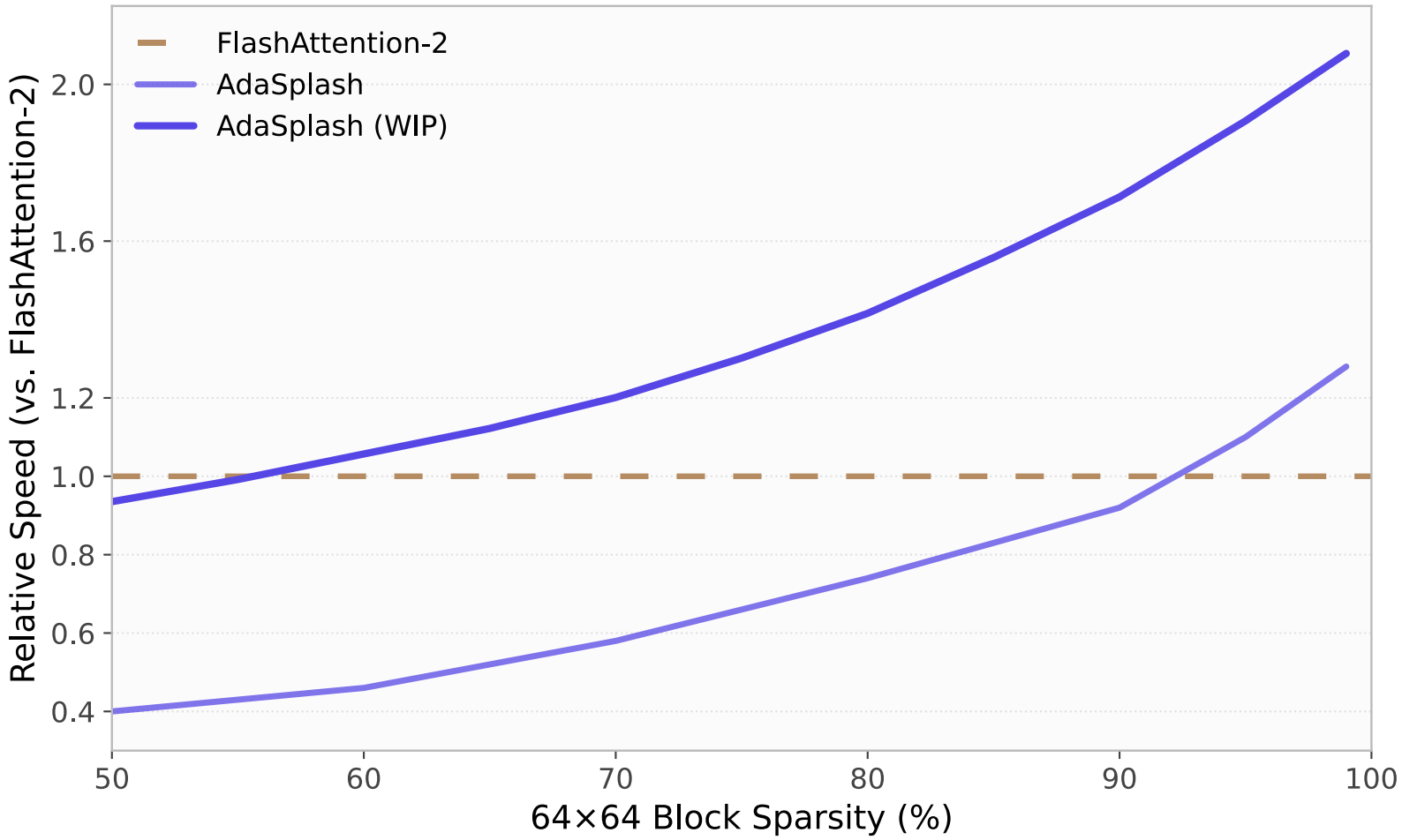
Interestingly, if $\text{sparsemax}(\boldsymbol{z})$ has already been evaluated (*i.e.*, in the forward step), then so has $S(\boldsymbol{z})$, hence the nonzeros of $\mathbf{J}_{\text{sparsemax}}(\boldsymbol{z}) \cdot \boldsymbol{v}$ can be computed in only $O(|S(\boldsymbol{z})|)$ time, which can be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

(Martins and Astudillo, 2016)

⚡ AdaSplash


FlashAttention-2
AdaSplash
Relative Speed (vs. FlashAttention-2)
2.0
1.6
1.2
1.0
0.8
0.6
0.4
50
60
70
80
90
100
64×64 Block Sparsity (%)

⚡ AdaSplash


FlashAttention-2
AdaSplash
AdaSplash (WIP)
Relative Speed (vs. FlashAttention-2)
2.0
1.6
1.2
1.0
0.8
0.6
0.4
50
60
70
80
90
100
64×64 Block Sparsity (%)

⚡AdaSplash

Table 1. Results for single-vector retrieval models on different tasks from the BEIR benchmark in terms of nDCG@10.

Model	Seq.	SciFact	NFC	FiQA	TREC-C
RoBERTa	512	51.7	23.1	27.8	60.1
RoBERTa ($\alpha = 1.5$)	512	50.8	24.2	27.6	71.0
RoBERTa ($\alpha = 2.0$)	512	52.2	23.8	25.7	65.5
ModernBERT	8192	57.7	22.4	25.7	67.6
ModernBERT ($\alpha = 1.5$)	8192	**58.4**	**25.7**	**29.6**	**75.2**
ModernBERT ($\alpha = 2.0$)	8192	58.0	25.4	29.3	71.1

8K context length

⚡AdaSplash

Table 1. Results for single-vector retrieval models on different tasks from the BEIR benchmark in terms of nDCG@10.

Model	Seq.	SciFact	NFC	FiQA	TREC-C
RoBERTa	512	51.7	23.1	27.8	60.1
RoBERTa ($\alpha = 1.5$)	512	50.8	24.2	27.6	71.0
RoBERTa ($\alpha = 2.0$)	512	52.2	23.8	25.7	65.5
ModernBERT	8192	57.7	22.4	25.7	67.6
ModernBERT ($\alpha = 1.5$)	8192	**58.4**	**25.7**	**29.6**	**75.2**
ModernBERT ($\alpha = 2.0$)	8192	58.0	25.4	29.3	71.1

8K context length

⚡AdaSplash

Table 1. Results for single-vector retrieval models on different tasks from the BEIR benchmark in terms of nDCG@10.

Model	Seq.	SciFact	NFC	FiQA	TREC-C
RoBERTa	512	51.7	23.1	27.8	60.1
RoBERTa ($\alpha = 1.5$)	512	50.8	24.2	27.6	71.0
RoBERTa ($\alpha = 2.0$)	512	52.2	23.8	25.7	65.5
ModernBERT	8192	57.7	22.4	25.7	67.6
ModernBERT ($\alpha = 1.5$)	8192	**58.4**	**25.7**	**29.6**	**75.2**
ModernBERT ($\alpha = 2.0$)	8192	58.0	25.4	29.3	71.1

8K context length

Table 3. Results on language modeling with GPT-2 in terms of final validation loss and accuracy on the HellaSwag task (Zellers et al., 2019), along with the average runtime per training step (in seconds) and peak memory usage (GB) per GPU.

Model	Val. Loss	HS Acc.	Runtime	Memory
GPT-2	3.283	30.4	**0.98**	**52.5**
GPT-2 ($\alpha = 1.5$)	**3.263**	**30.6**	1.03	**52.5**
w/ Torch sorting	-	-	3.61	73.8
w/ Torch bisection	-	-	7.78	77.6

1K context length

⚡AdaSplash

Table 1. Results for single-vector retrieval models on different tasks from the BEIR benchmark in terms of nDCG@10.

Model	Seq.	SciFact	NFC	FiQA	TREC-C
RoBERTa	512	51.7	23.1	27.8	60.1
RoBERTa ($\alpha = 1.5$)	512	50.8	24.2	27.6	71.0
RoBERTa ($\alpha = 2.0$)	512	52.2	23.8	25.7	65.5
ModernBERT	8192	57.7	22.4	25.7	67.6
ModernBERT ($\alpha = 1.5$)	8192	**58.4**	**25.7**	**29.6**	**75.2**
ModernBERT ($\alpha = 2.0$)	8192	58.0	25.4	29.3	71.1

8K context length

Table 3. Results on language modeling with GPT-2 in terms of final validation loss and accuracy on the HellaSwag task (Zellers et al., 2019), along with the average runtime per training step (in seconds) and peak memory usage (GB) per GPU.

Model	Val. Loss	HS Acc.	Runtime	Memory
GPT-2	3.283	30.4	**0.98**	**52.5**
GPT-2 ($\alpha = 1.5$)	**3.263**	**30.6**	1.03	**52.5**
w/ Torch sorting	-	-	3.61	73.8
w/ Torch bisection	-	-	7.78	77.6

1K context length

Sparsity of α-entmax attention

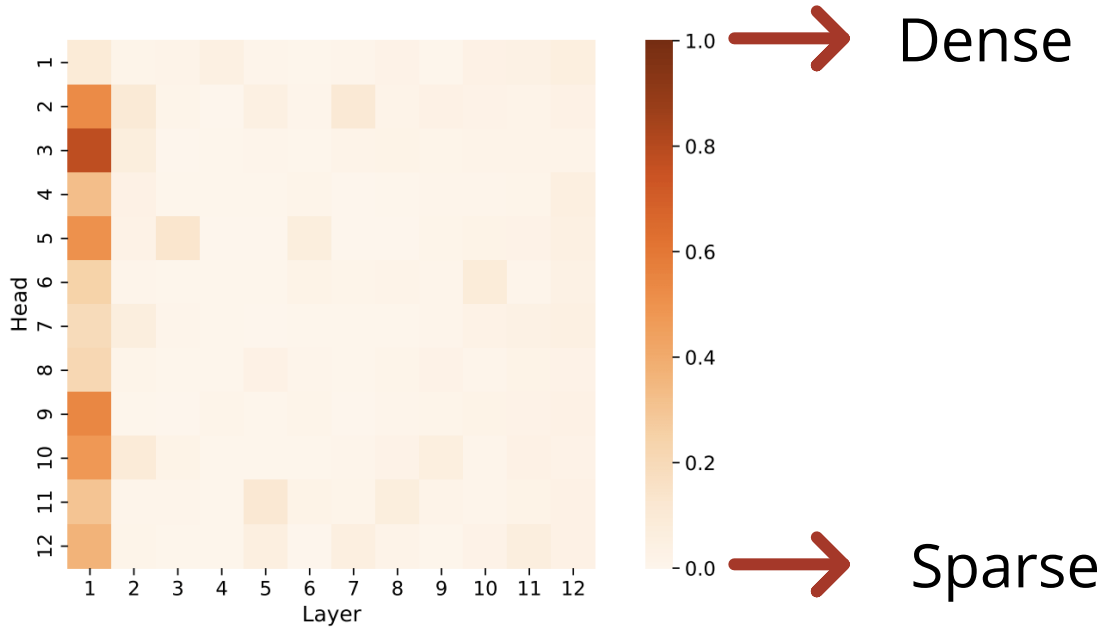
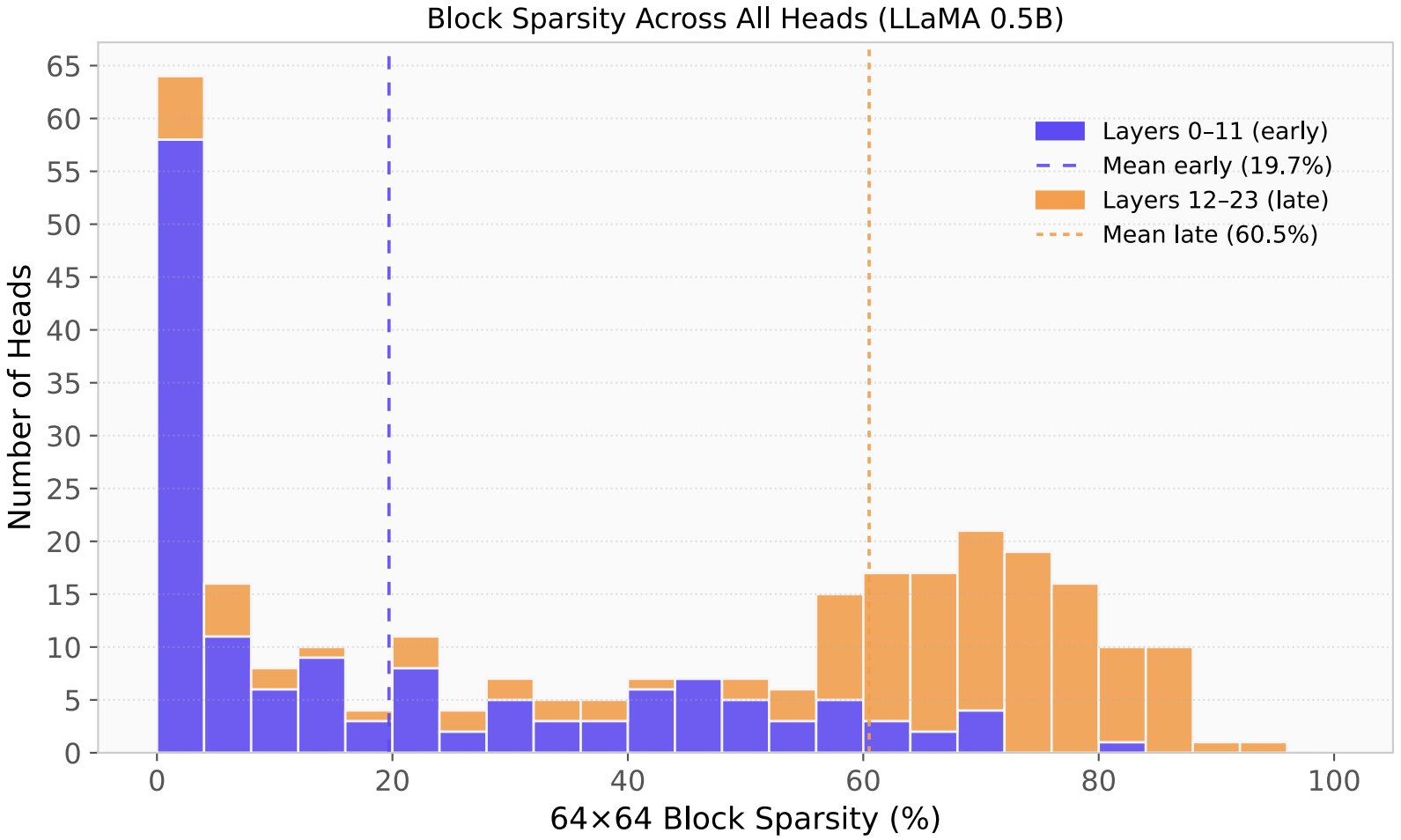


Figure 4. Ratio of non-zero attention scores for GPT-2 ($\alpha = 1.5$).

AdaSplash Sparsity



Final Thoughts

- Strong generalization
  - outperforms Mamba-3
- On par, or better than softmax
- Triton, not CUDA
- Academic budget
- v1

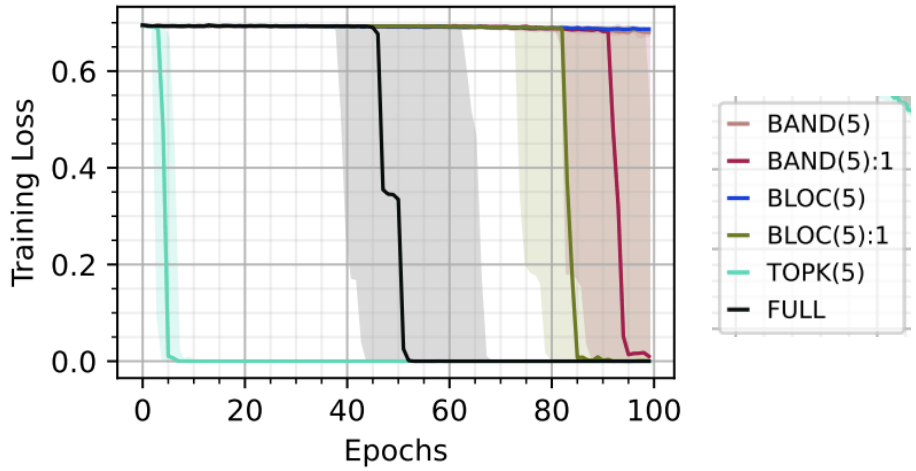
Future Directions

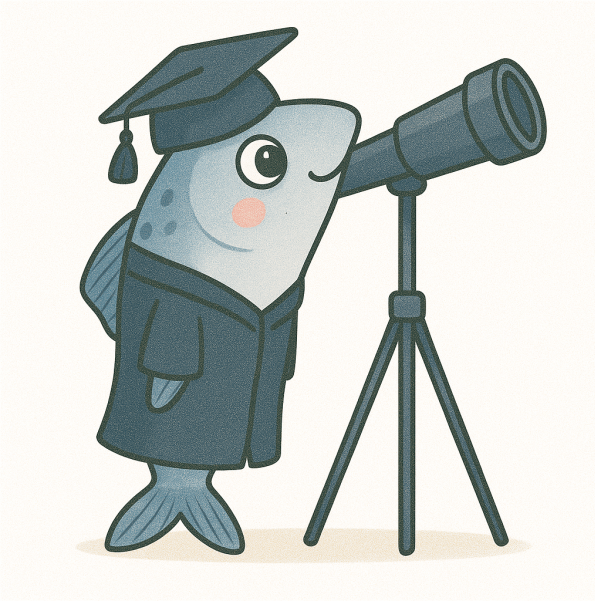
- Can we learn faster with sparse attention? ⚡
  - top-k attention apparently does [1]
  - data efficiency

- Mix dense and sparse attention 👨‍🍳
  - Early layers are dense
  - Final layers are sparse

- AdaPrefill and AdaDecoding 👨‍🍳

- New AdaSplash versions 👨‍🍳
  - Faster, better, more optimized
  - Stay tuned!





[1] Transformers Learn Faster with Semantic Focus. Ram et al., 2025

THANK YOU FOR YOUR ATTENTION

Extra slides

α-entmax Attention

$$\alpha\text{-entmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

Extensively tested on many tasks and domains!

- Interpretability [1, 2, 3, ...]
- Machine Translation [1, 2, 3, ...]
- Language Modeling [1, 2, 3, ...]
- Retrieval and Memory [1, 2, 3]
- Emergent Communication [1, 2]
- Conformal Prediction [1]

α-entmax Attention

$$\alpha\text{-entmax}(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$$

Extensively tested on many tasks and domains!

- Interpretability [1, 2, 3, ...]
- Machine Translation [1, 2, 3, ...]
- Language Modeling [1, 2, 3, ...]
- Retrieval and Memory [1, 2, 3]
- Emergent Communication [1, 2]
- Conformal Prediction [1]

usually

$$\alpha = 1.5 \;\geq\; \alpha = 2.0 \;\approx\; \alpha = 1.0\ (\text{softmax})$$

Sparsemax Attention ($\alpha = 2$)

$\mathcal{S}_i = \{j \in [n] : p_{ij} > 0\}, \quad \tau_i = \tau(\boldsymbol{K}\boldsymbol{q}_i)$

$$
\begin{aligned}
\boldsymbol{y}_i &= \sum_{j=1}^{n} \max\left(\boldsymbol{q}_i^\top \boldsymbol{k}_j - \tau_i, 0\right) \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \left(\boldsymbol{q}_i^\top \boldsymbol{k}_j - \tau_i\right) \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \left(\boldsymbol{q}_i^\top \boldsymbol{k}_j\right) \boldsymbol{v}_j - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \left(\boldsymbol{k}_j^\top \boldsymbol{q}_i\right) - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \\
&= \left(\sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \boldsymbol{k}_j^\top\right) \boldsymbol{q}_i - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j
\end{aligned}
$$

Sparsemax Attention ($\alpha = 2$)

$\mathcal{S}_i = \{j \in [n] : p_{ij} > 0\}, \quad \tau_i = \tau(\boldsymbol{K}\boldsymbol{q}_i)$

$$
\begin{aligned}
\boldsymbol{y}_i &= \sum_{j=1}^{n} \max\left(\boldsymbol{q}_i^\top \boldsymbol{k}_j - \tau_i, 0\right) \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \left(\boldsymbol{q}_i^\top \boldsymbol{k}_j - \tau_i\right) \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \left(\boldsymbol{q}_i^\top \boldsymbol{k}_j\right) \boldsymbol{v}_j - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \\
&= \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \left(\boldsymbol{k}_j^\top \boldsymbol{q}_i\right) - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \\
&= \left(\sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j \boldsymbol{k}_j^\top\right) \boldsymbol{q}_i - \tau_i \sum_{j \in \mathcal{S}_i} \boldsymbol{v}_j
\end{aligned}
$$

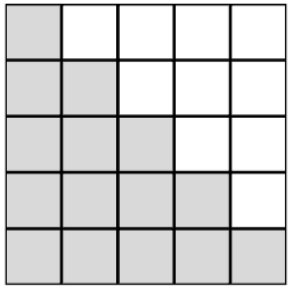
vectorized format:

$$
D_{ij} = \begin{cases} 1, & \text{if } j \in \mathcal{S}_i \\ 0, & \text{otherwise} \end{cases}
$$

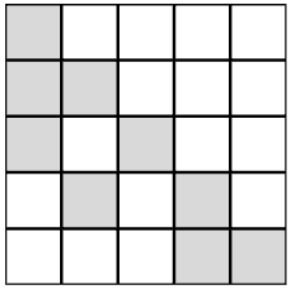
$$
\begin{aligned}
\boldsymbol{Y} &= (\boldsymbol{D} \odot \boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V} - \operatorname{diag}(\boldsymbol{\tau})\boldsymbol{D}\boldsymbol{V} \\
&= \left(\boldsymbol{D} \odot \boldsymbol{Q}\boldsymbol{K}^\top - \boldsymbol{D} \odot \boldsymbol{\tau}\mathbf{1}^\top\right)\boldsymbol{V} \\
&= \left(\boldsymbol{D} \odot (\boldsymbol{Q}\boldsymbol{K}^\top - \boldsymbol{\tau}\mathbf{1}^\top)\right)\boldsymbol{V} \\
&= \left(\boldsymbol{D} \odot (\bar{\boldsymbol{Q}}\bar{\boldsymbol{K}}^\top)\right)\boldsymbol{V}
\end{aligned}
$$

Sparsemax Attention ($\alpha = 2$)

$$\boldsymbol{Y} = \left(\boldsymbol{L} \odot (\boldsymbol{Q}\boldsymbol{K}^\top)\right)\boldsymbol{V}$$

$\boldsymbol{L} =$ 

$$\boldsymbol{Y} = \left(\boldsymbol{D} \odot (\bar{\boldsymbol{Q}}\bar{\boldsymbol{K}}^\top)\right)\boldsymbol{V}$$

$\boldsymbol{D} =$ 

Within an active support region (fixed D),

sparsemax attention is akin to **selective** and **normalized** linear attention!

What about 1.5-entmax? Mixture of first- and second-order interactions!

=> selective and normalized **quadratic** attention

Self-Attention

query

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

keys

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

values

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\text{score}(\boldsymbol{Q}, \boldsymbol{K}))\boldsymbol{V}$$

Self-Attention

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

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

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\text{score}(\boldsymbol{Q}, \boldsymbol{K}))\boldsymbol{V}$$

1. Compute pairwise scores $\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$

Self-Attention

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

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

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\text{score}(\boldsymbol{Q}, \boldsymbol{K}))\boldsymbol{V}$$

1. Compute pairwise scores $\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$
2. Map rows to probabilities $\boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n}$

Self-Attention

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

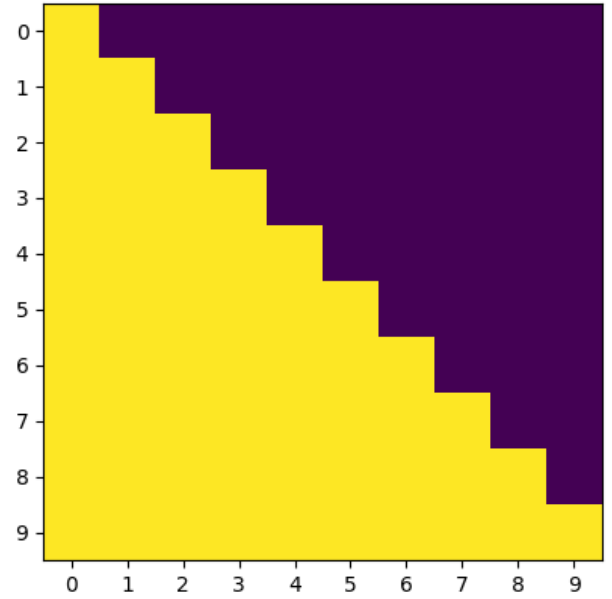
keys $\boldsymbol{K} \in \mathbb{R}^{n \times d}$

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\text{score}(\boldsymbol{Q}, \boldsymbol{K}))\boldsymbol{V}$$

1. Compute pairwise scores $\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$
2. Map rows to probabilities $\boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n}$

causal masking $\boldsymbol{Z} = \boldsymbol{M} \odot \boldsymbol{Z} + \neg \boldsymbol{M} \odot (-\infty_{n \times n})$



Self-Attention

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

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

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

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \pi(\text{score}(\boldsymbol{Q}, \boldsymbol{K}))\boldsymbol{V}$$

1. Compute pairwise scores $\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$
2. Map rows to probabilities $\boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n}$

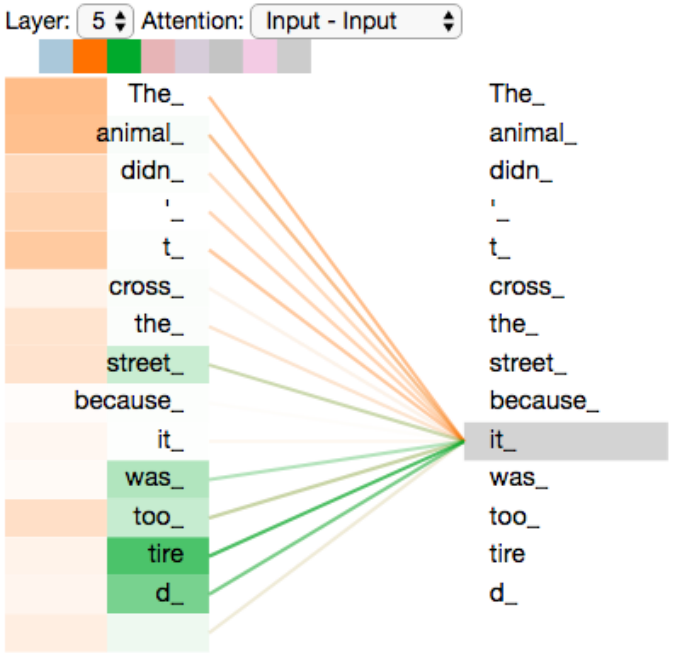
$$\text{softmax}(\boldsymbol{z})_j = \exp(\boldsymbol{z}_j) / \sum_k \exp(\boldsymbol{z}_k)$$

Multi-head Attention

Multiple attention heads allow the model to **jointly attend to different representation subspaces**

$$\text{multihead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}_O$$

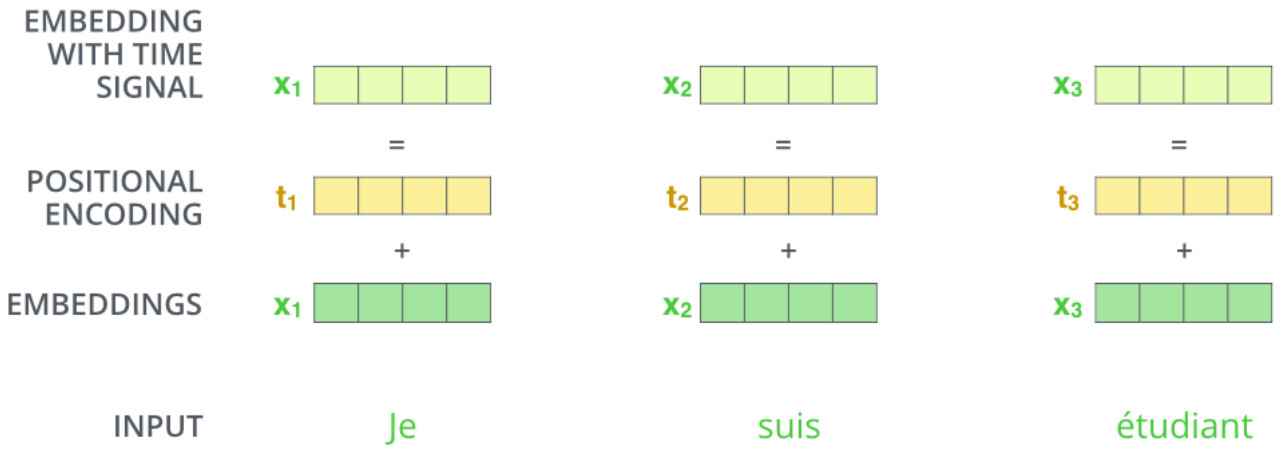
$$\text{head}_i = \text{attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V)$$



Different heads often specialize to focus on different patterns!

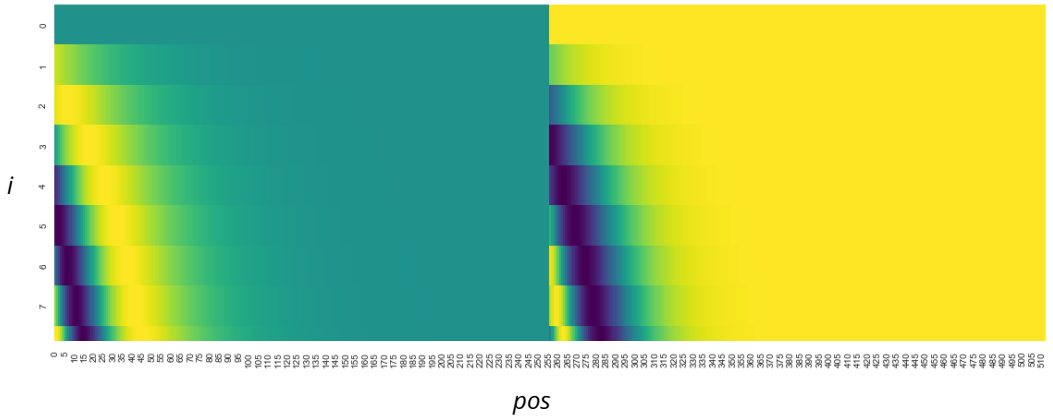
Absolute Positional Encoding

Attention is inherently **order-agnostic.** We need to add position information!



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

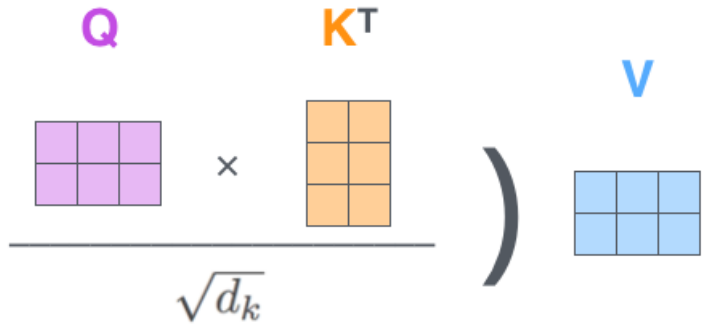


The Core of Transformers

Self-attention allows tokens $\langle \boldsymbol{x}_1, ..., \boldsymbol{x}_n \rangle$ to attend to **each other**

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{n \times d}$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{n \times d}$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{n \times d}$$

$$\text{softmax}\left(\frac{\boldsymbol{Q} \times \boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}$$

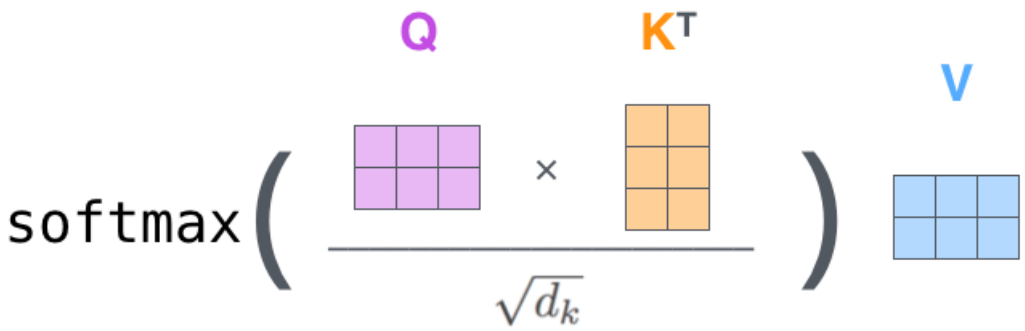


$$\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$$
$$\boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n}$$
$$\boldsymbol{O} = \boldsymbol{P}\boldsymbol{V} \in \mathbb{R}^{n \times d}$$

quadractic complexity

The Core of Transformers

Self-attention allows tokens $\langle \boldsymbol{x}_1, ..., \boldsymbol{x}_n \rangle$ to attend to **each other**



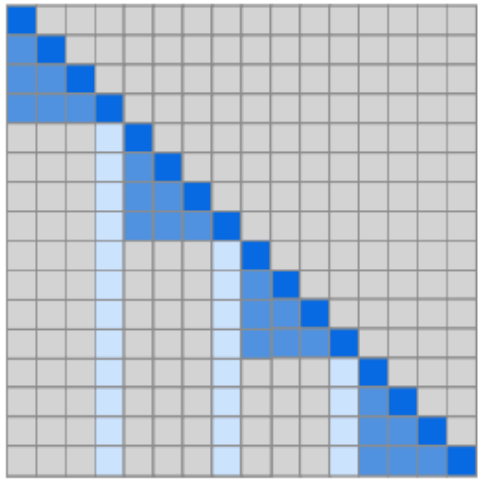
$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{n \times d}$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{n \times d}$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{n \times d}$$

$$\boldsymbol{Z} = \text{score}(\boldsymbol{Q}, \boldsymbol{K}) \in \mathbb{R}^{n \times n}$$
$$\boldsymbol{P} = \pi(\boldsymbol{Z}) \in \mathbb{R}^{n \times n}$$
$$\boldsymbol{O} = \boldsymbol{P}\boldsymbol{V} \in \mathbb{R}^{n \times d}$$

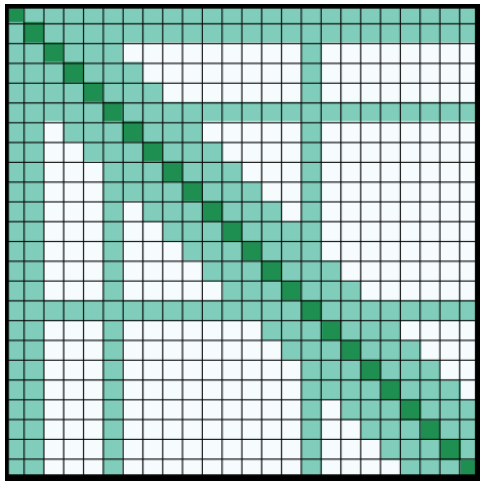
Sparse Attention

Method	Adaptivity	Flexibility	Differentiability	Time Cost	Memory Cost
Fixed Pattern Methods					
Window Attention (Longformer, Sparse Transformer)	Low	Low	Full	$O(n \times w)$	$O(n \times w)$
Global Attention (Longformer)	Low	Low	Full	$O(n \times g)$	$O(n \times g)$
Random Attention (BigBird)	Low	Low	Full	$O(n \times (w+g+r))$	$O(n \times (w+g+r))$
Dynamic Token Selection Methods					
Top-k	High	Medium	None	$O(n \log n)$	$O(n \times k)$
LSH-based (Reformer)	High	Medium	Approx	$O(n \log n)$	$O(n \log n)$
Clustering (Routing Transformer)	High	Medium	Approx	$O(n\sqrt{n})$	$O(n \times c)$
Probability					
α-entmax	High	High	Full	$O(n^2 \log n)$	$O(n^2)$

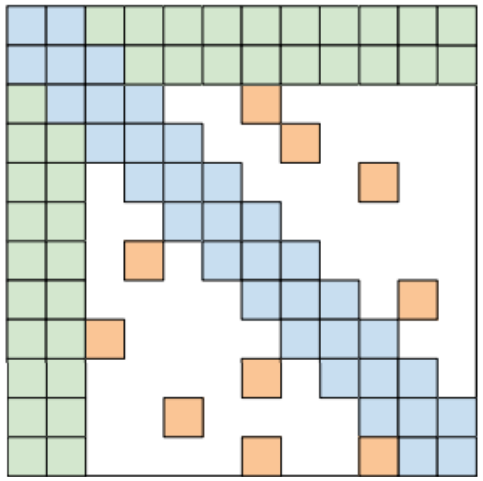
Efficiency



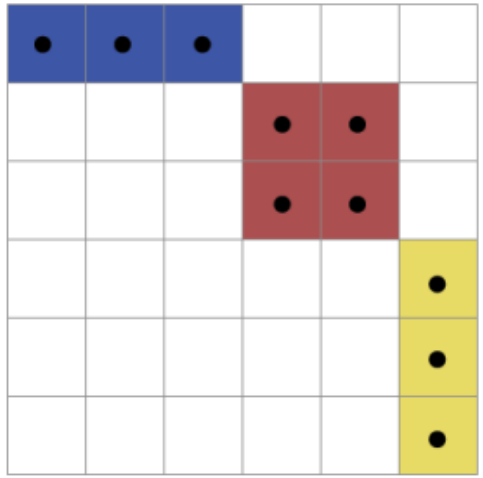
Sparse transformer



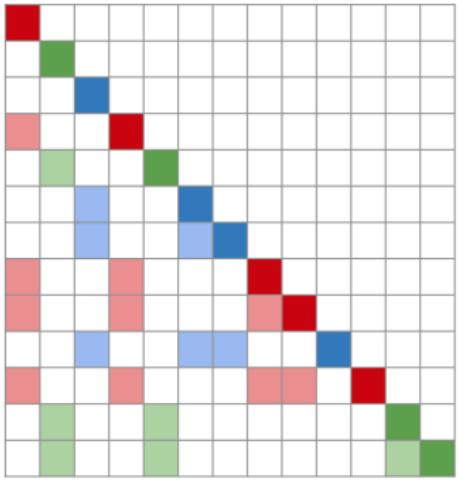
Longformer



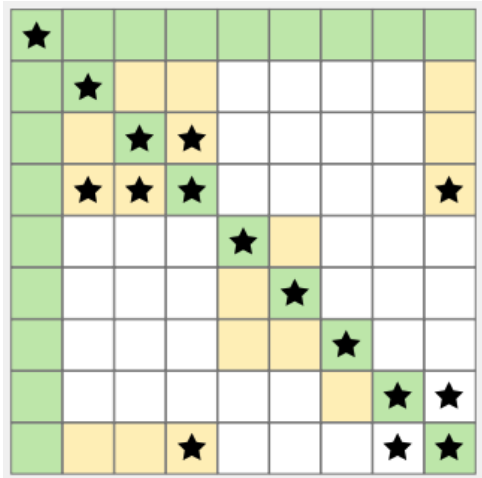
BigBird



Reformer

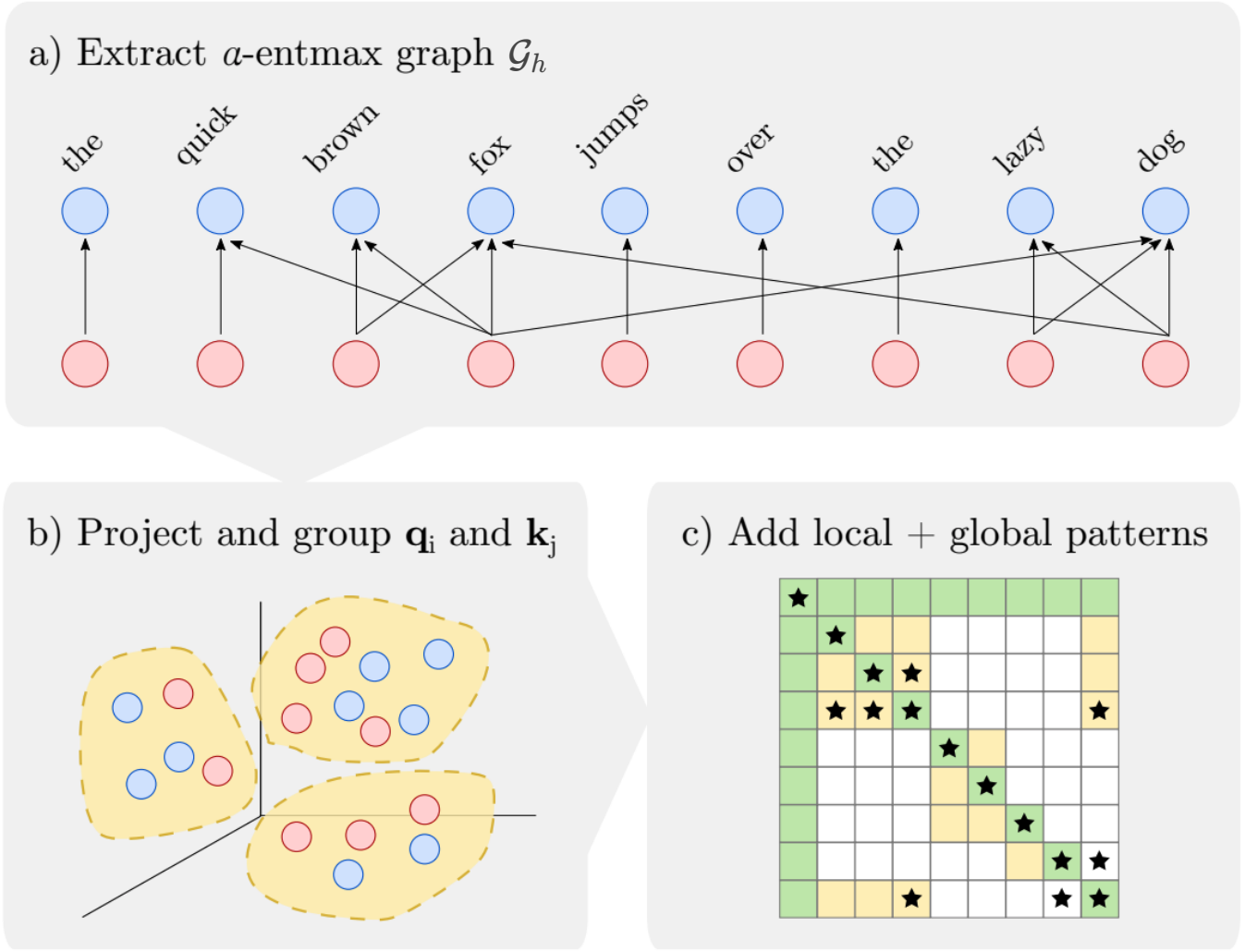


Routing transformer



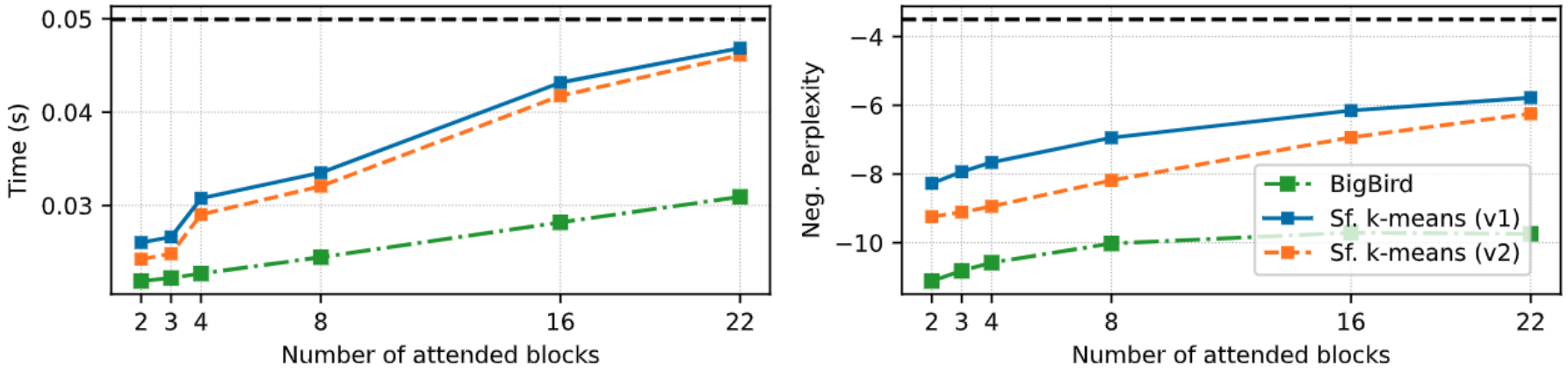
Our approach

Sparsefinder



Efficient Sparsefinder

- Learn projections of contiguous-chunked tokens: **block-sparsity**



v1: top-*k* queries and keys closest to each centroid

v2: top-*k* queries for each clusters *x* top-*k* keys for each cluster

k = number of attended blocks

window size = 3