

Rethinking Long-Context Modeling with Efficient Sparse Attention

Marcos V. Treviso
IST, University of Lisbon

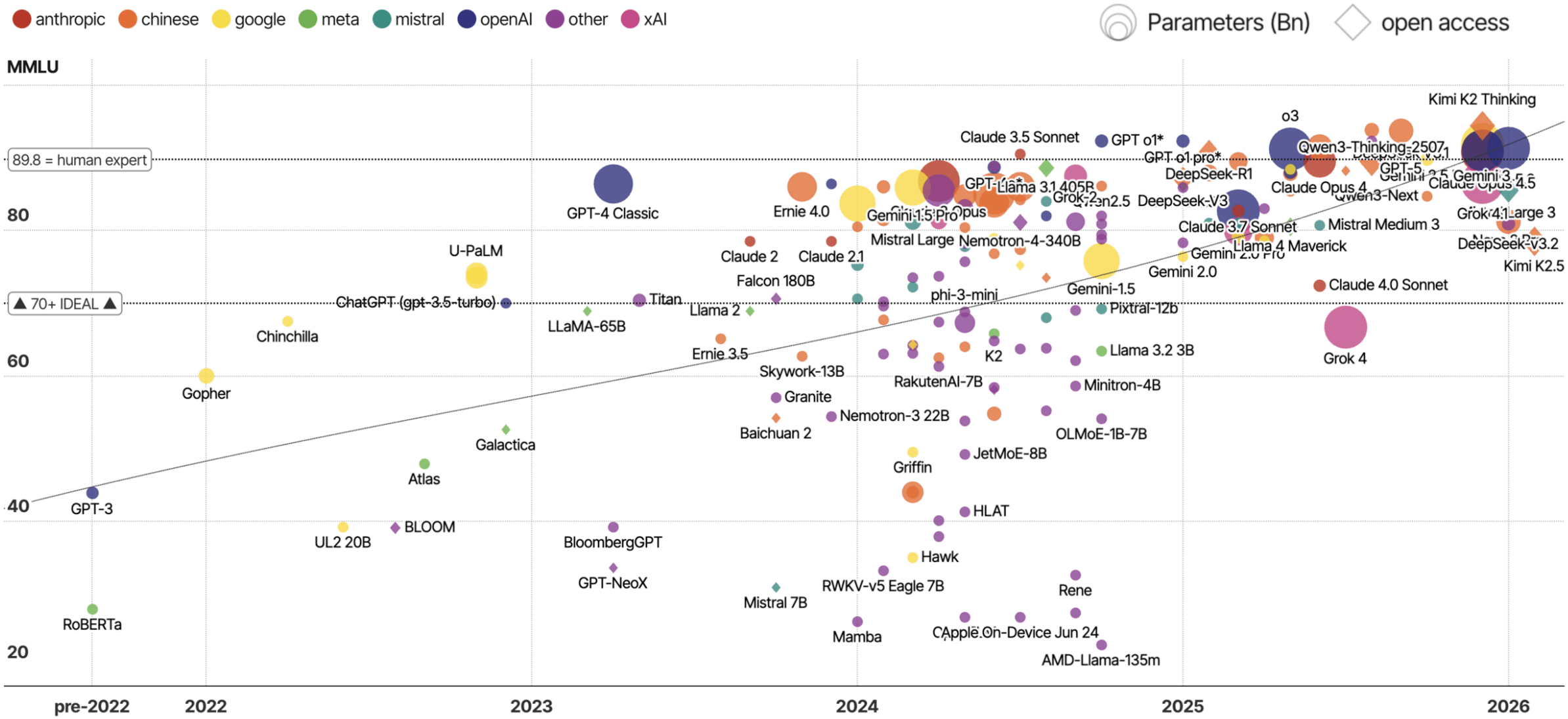
October 20, 2026

BRACIS - ENIAC - STIL 2017

~word embeddings~

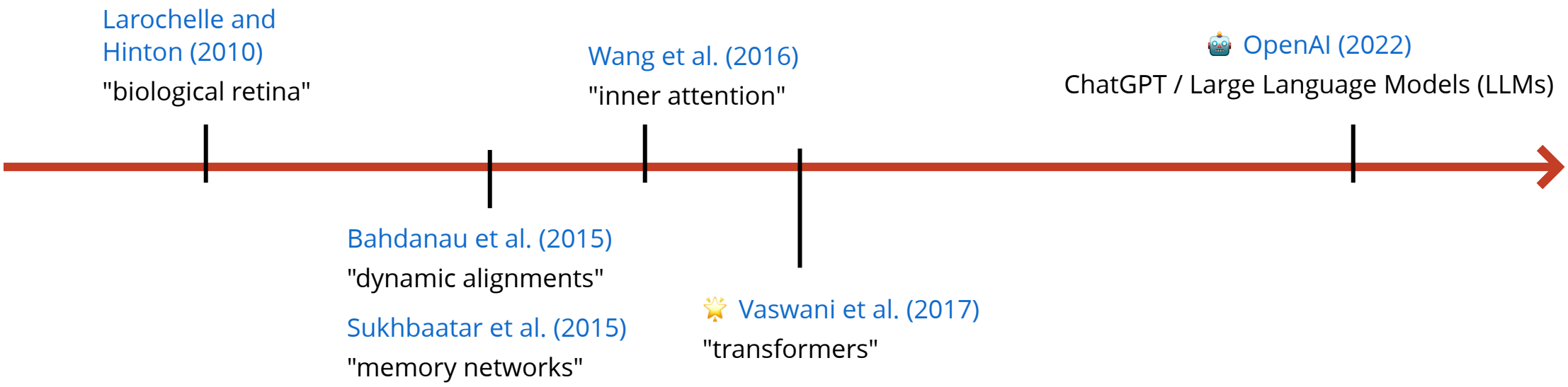


Models keep getting better, and bigger

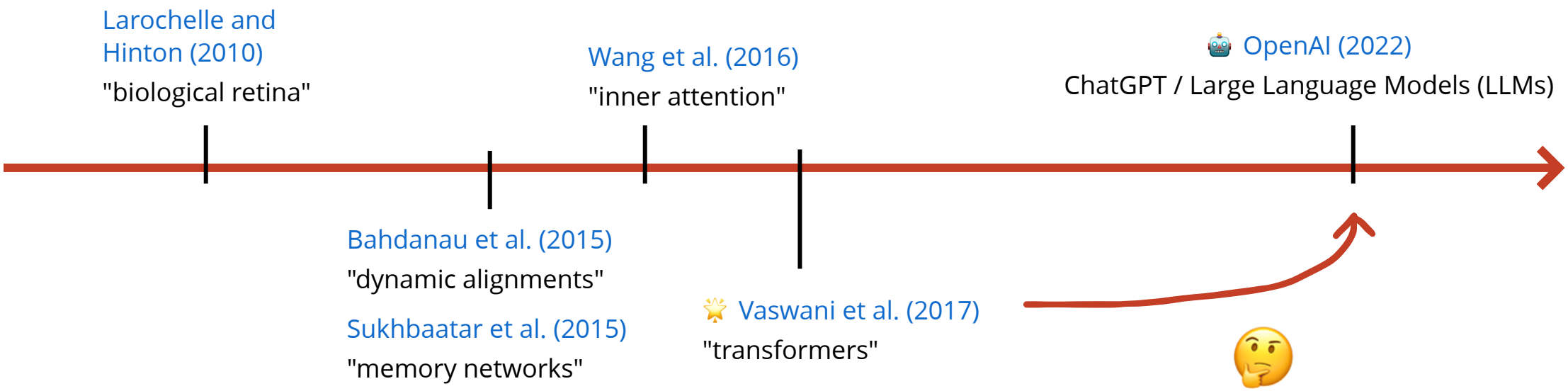


[source]

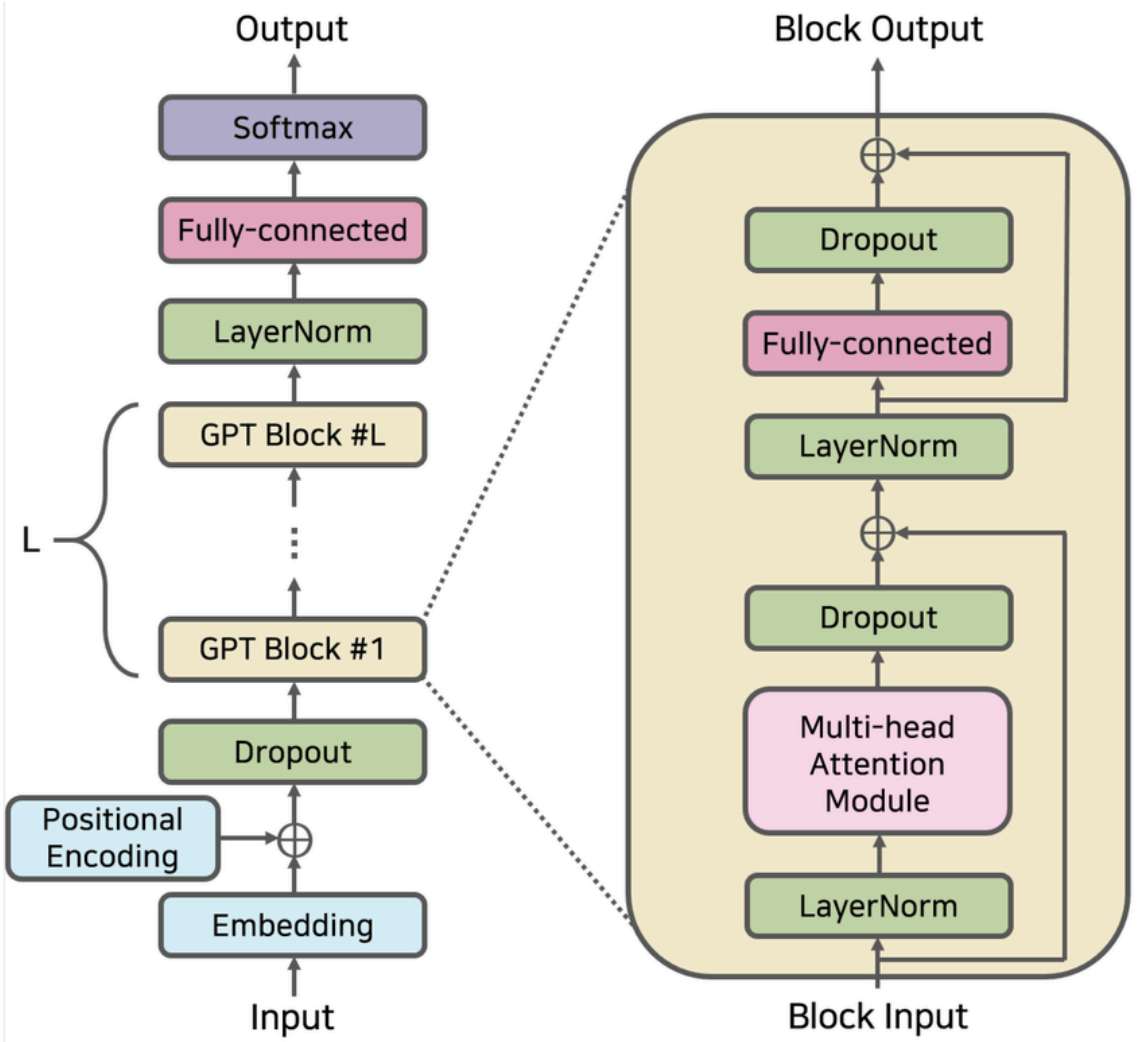
Large Language Models (LLMs)



Large Language Models (LLMs)

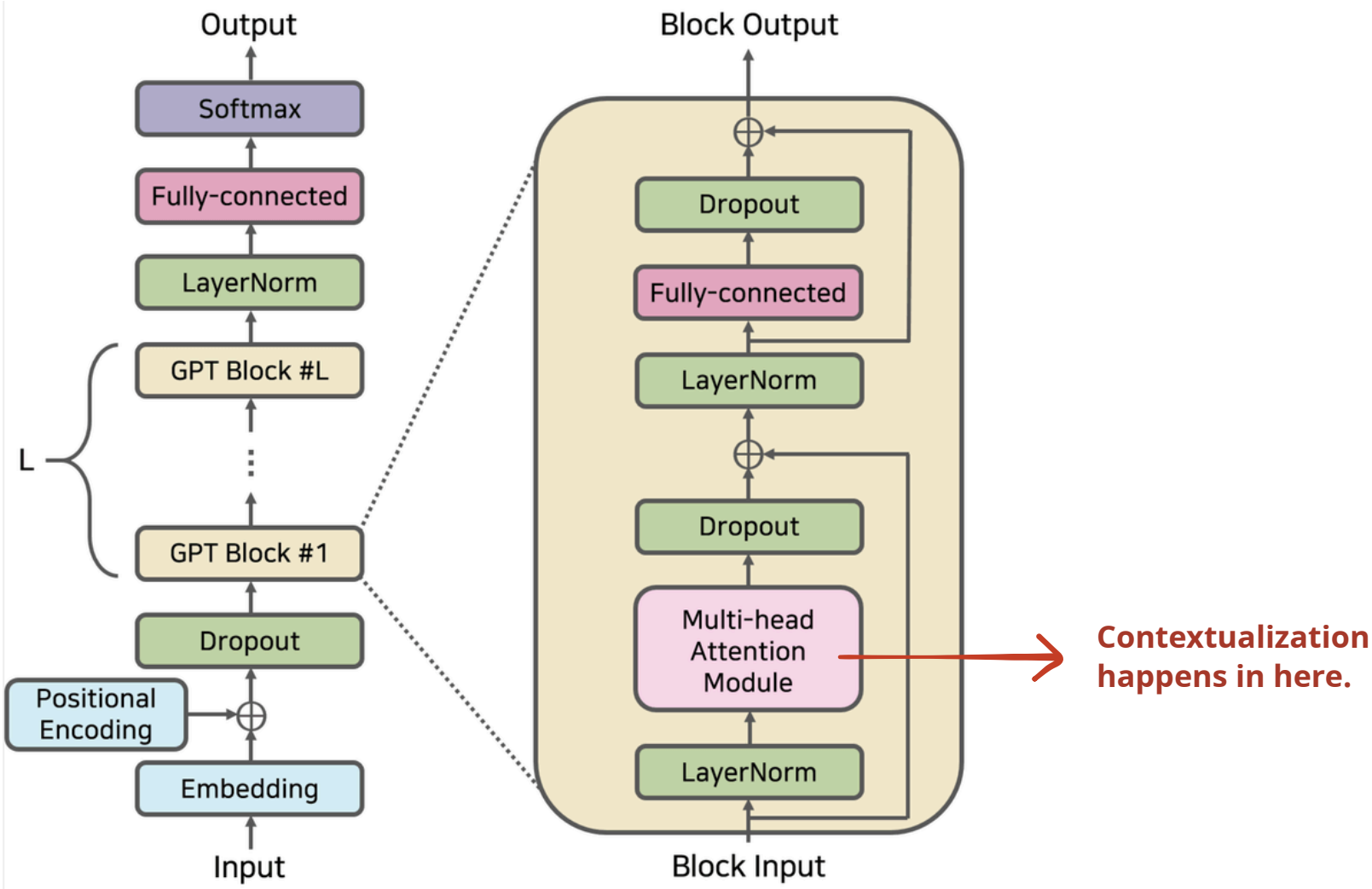


Transformers (GPT)



Language models are unsupervised multitask learners. Radford and Wu et al. (2022)

Transformers (GPT)



Language models are unsupervised multitask learners. Radford and Wu et al. (2022)

Self-Attention

queries

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

keys

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

values

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

1. Compute pairwise scores

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

2. Map rows to probabilities

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

3. Convex combination

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

Self-Attention

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

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

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

1. Compute pairwise scores

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

2. Map rows to probabilities

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

3. Convex combination

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

```
1 Z = torch.matmul(Q, K.transpose(-1, -2)) / d**0.5
2 P = torch.softmax(Z, dim=-1)
3 O = torch.matmul(P, V)
```

Self-Attention

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

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

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

1. Compute pairwise scores

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

2. Map rows to probabilities

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

3. Convex combination

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

```
Z = torch.matmul(Q, K.transpose(-1, -2)) / d**0.5
P = torch.softmax(Z, dim=-1)
O = torch.matmul(P, V)
```

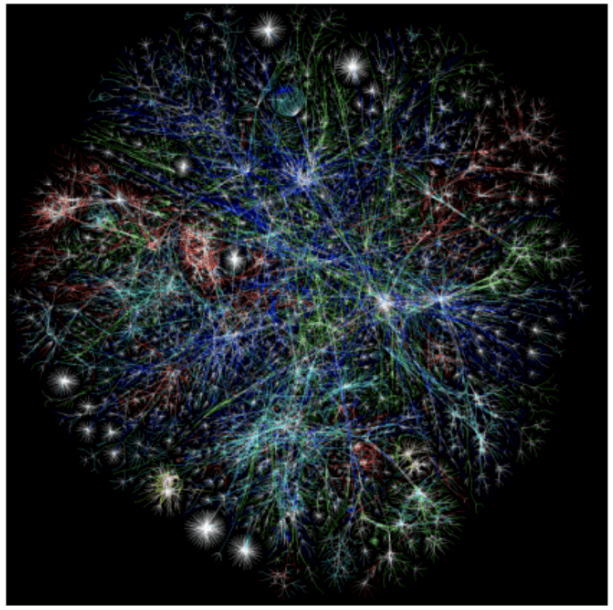
$B = 1$ sample
$n = 4096$ tokens
$H = 16$ heads
32 float precision

$4096^2 \times 16 \times 32/8$ = **1GB**

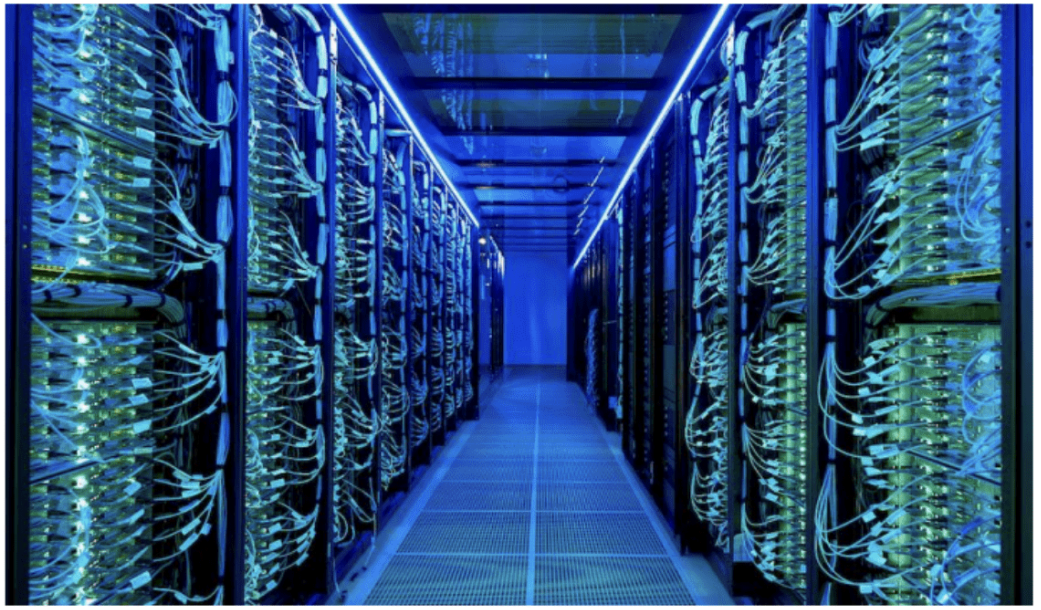
for a single sample
with just 4K tokens!

+ model params and optimizer states

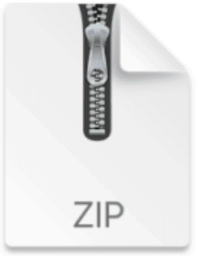
Scaling is challenging



Chunk of the internet,
~10TB of text



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



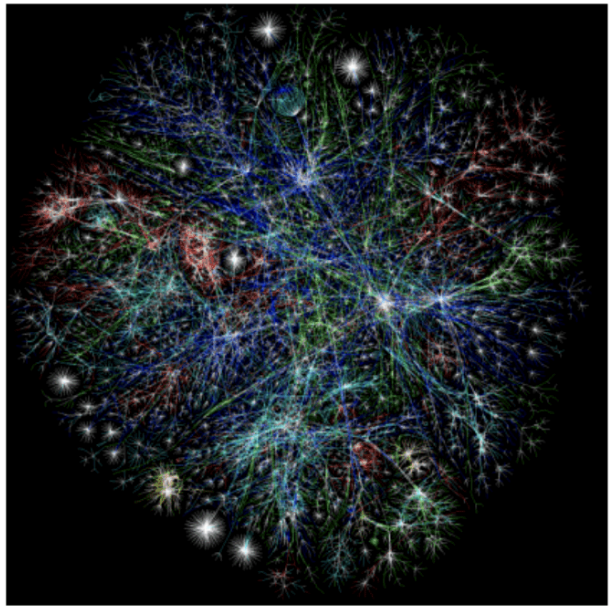
parameters.zip

~140GB file

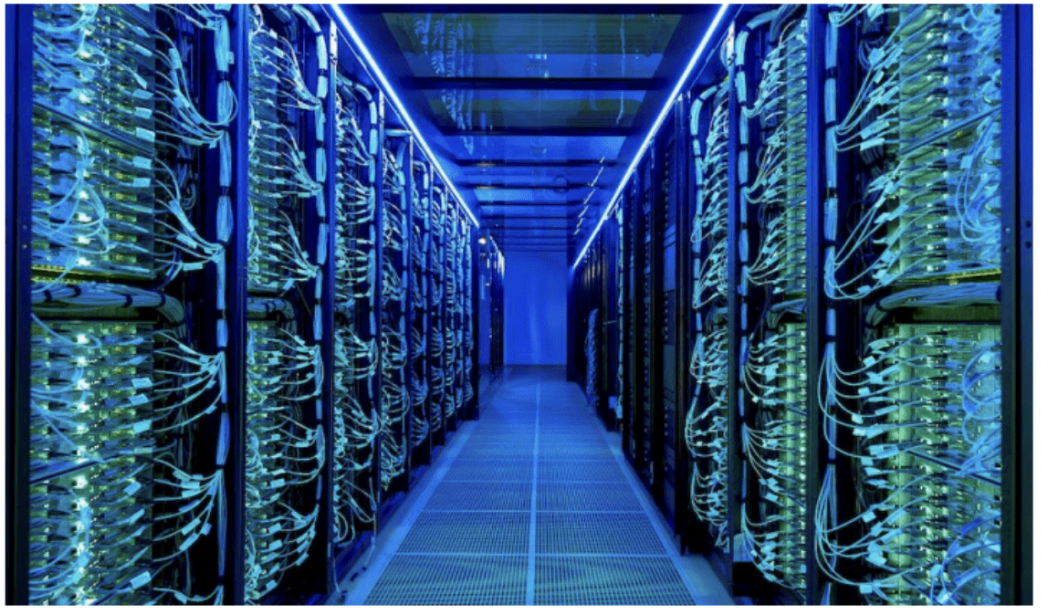
* Numbers for LLaMA-2 70B

Slide by Andrej Karpathy

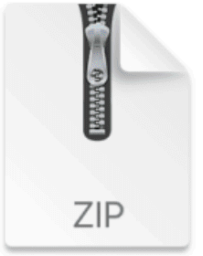
Scaling is challenging



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** with a context length of **4096 tokens***

* Numbers for LLaMA-2 70B

Slide by Andrej Karpathy

📄 Text

$10^3 - 10^5$

paper ≈ 10K · book ≈ 150K · codebase ≈ millions

🎧 Audio

$10^5 - 10^6$

1 hour ≈ 100K–500K tokens, codec-dependent

🎬 Video

$10^6 - 10^7$

1 hour @ 1 fps × ~250 tokens/frame ≈ 1M

🤖 Agents

∞

unbounded... context is the working memory

📄 **Text** $10^3 - 10^5$

paper ≈ 10K · book ≈ 150K · codebase ≈ millions

🎧 **Audio** $10^5 - 10^6$

1 hour ≈ 100K–500K tokens, codec-dependent

🎬 **Video** $10^6 - 10^7$

1 hour @ 1 fps × ~250 tokens/frame ≈ 1M

🤖 **Agents** ∞

unbounded... context is the working memory

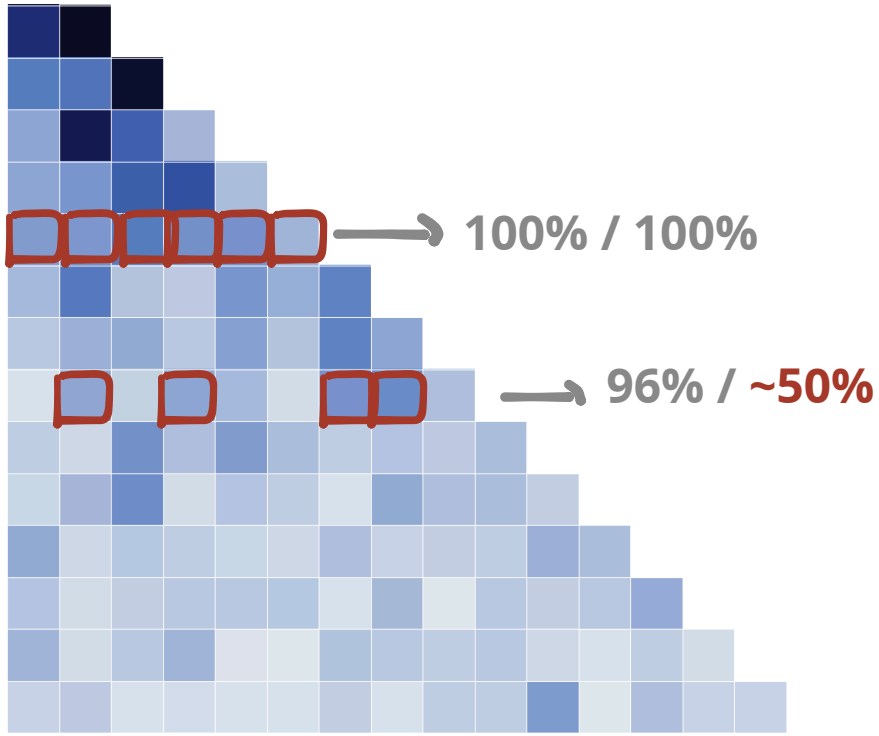
If we want models to handle **long context**,
we will need some kind of **efficient attention**

Quick quiz



A LLaMA-3 (8B) reads a prompt with **128K tokens**.

On average, **what fraction** of the entries is enough to capture **96% of all attention mass**?



A ~50%

B ~25%

C ~3%

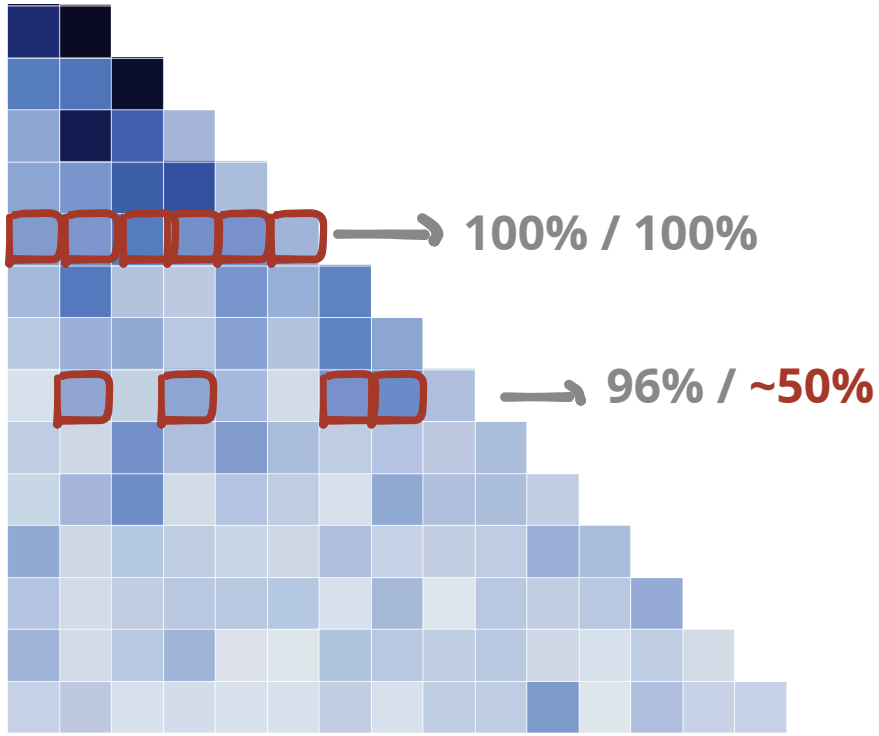
D ~0.01%

Quick quiz



A LLaMA-3 (8B) reads a prompt with **128K tokens**.

On average, **what fraction** of the entries is enough to capture **96% of all attention mass**?



A ~50%

B ~25%

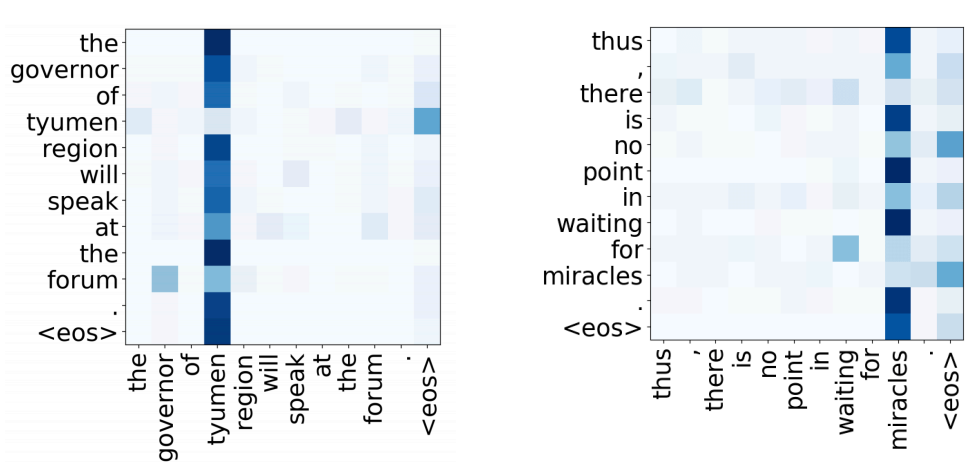
C ~3%

D ~0.01%

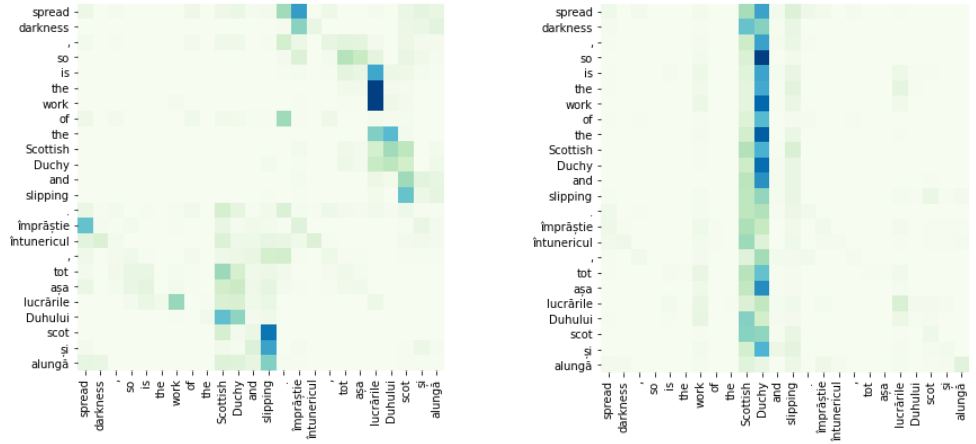
StreamingLLM: Efficient Streaming Language Models with Attention Sinks. Xiao et al. ICLR 2024

Attention heads are very sparse...

...and we've known this for a long time!



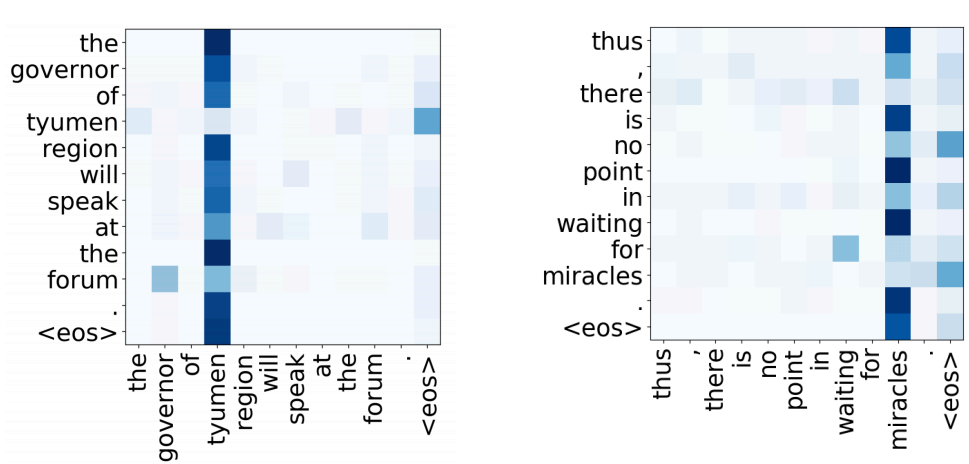
(Voita et al., 2019)



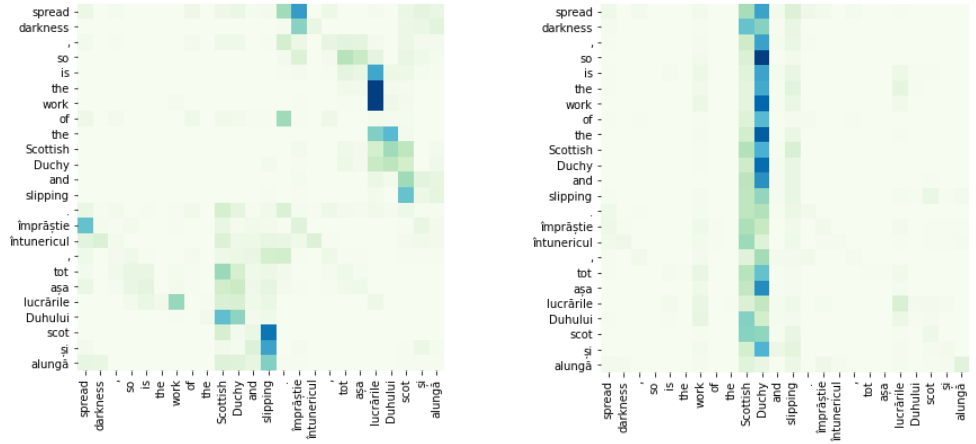
(Correia et al., 2019)

Attention heads are very sparse...

...and we've known this for a long time!



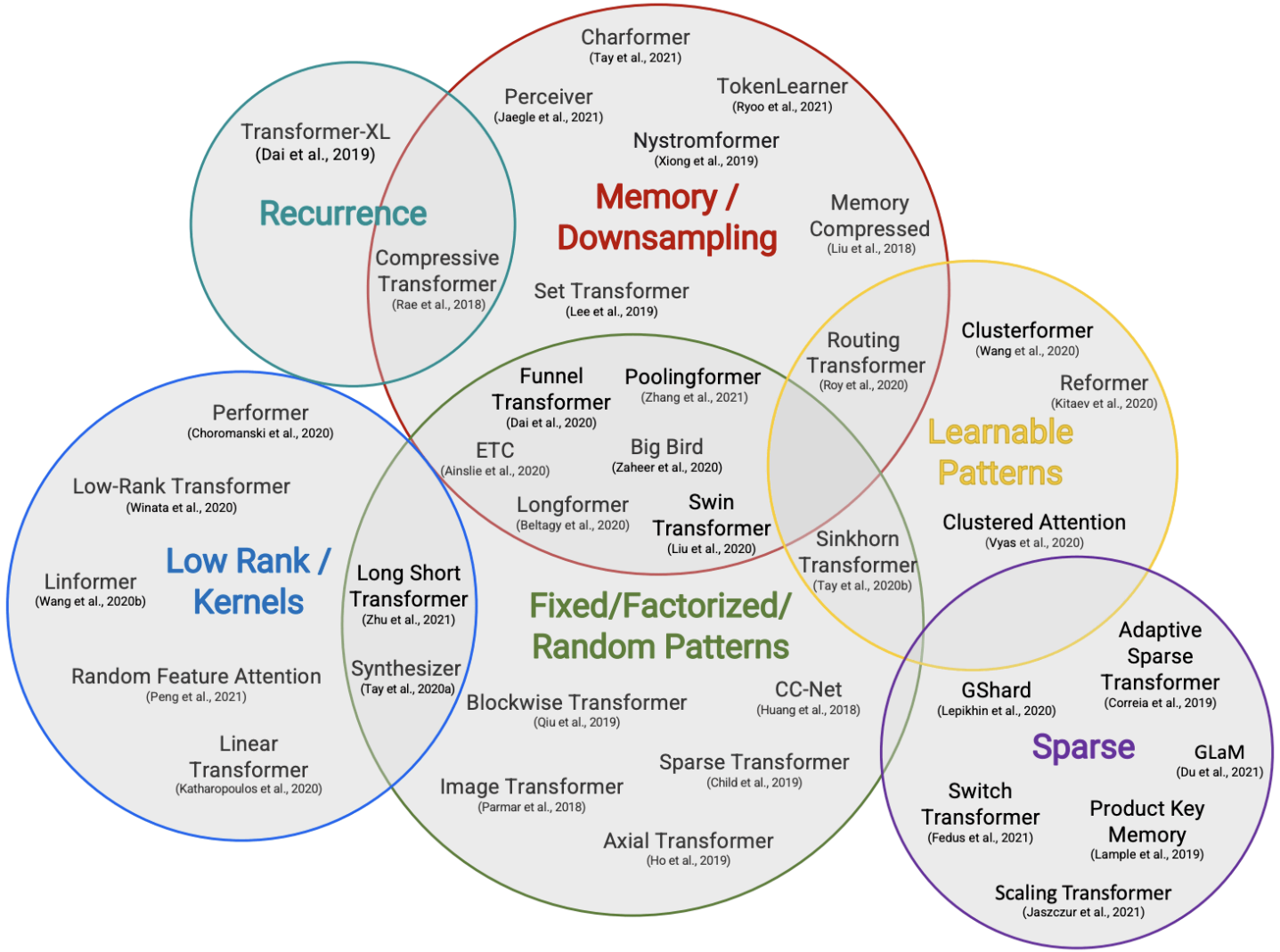
(Voita et al., 2019)



(Correia et al., 2019)

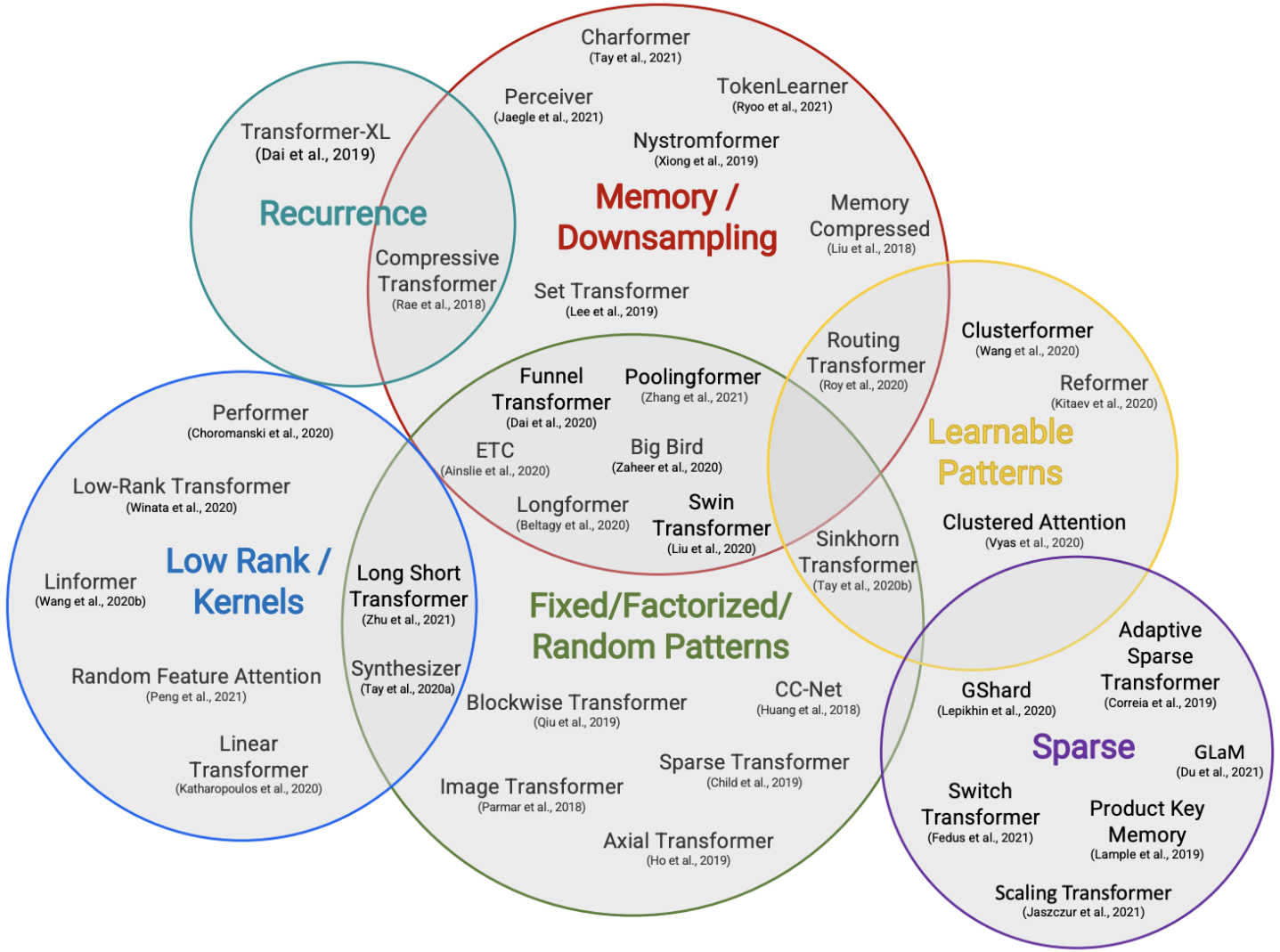
yet, turning **interpretability** into **efficiency** is not trivial.

Efficient Transformers



Efficient Transformers: A Survey. Tay et al. (2020)

Efficient Transformers

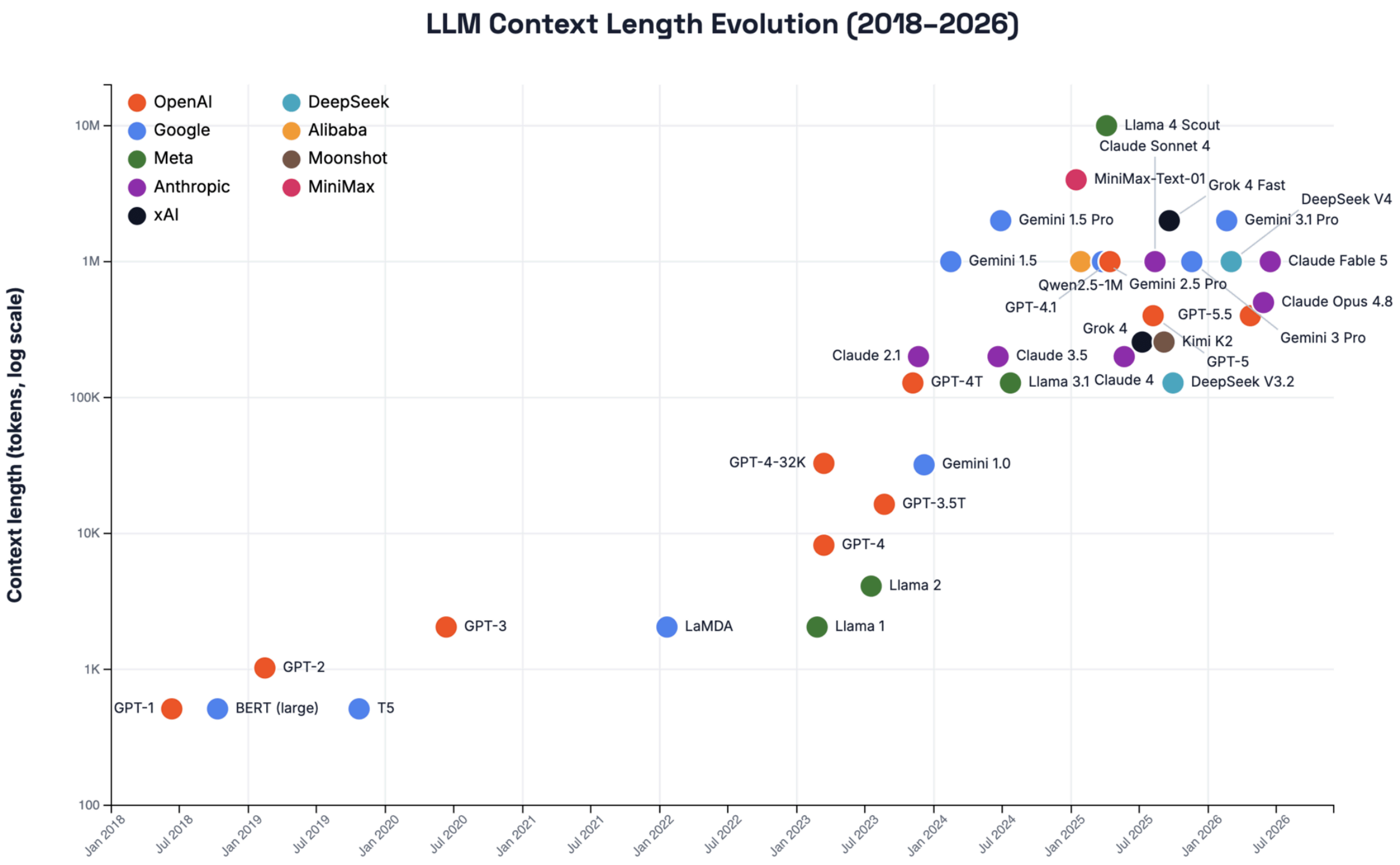


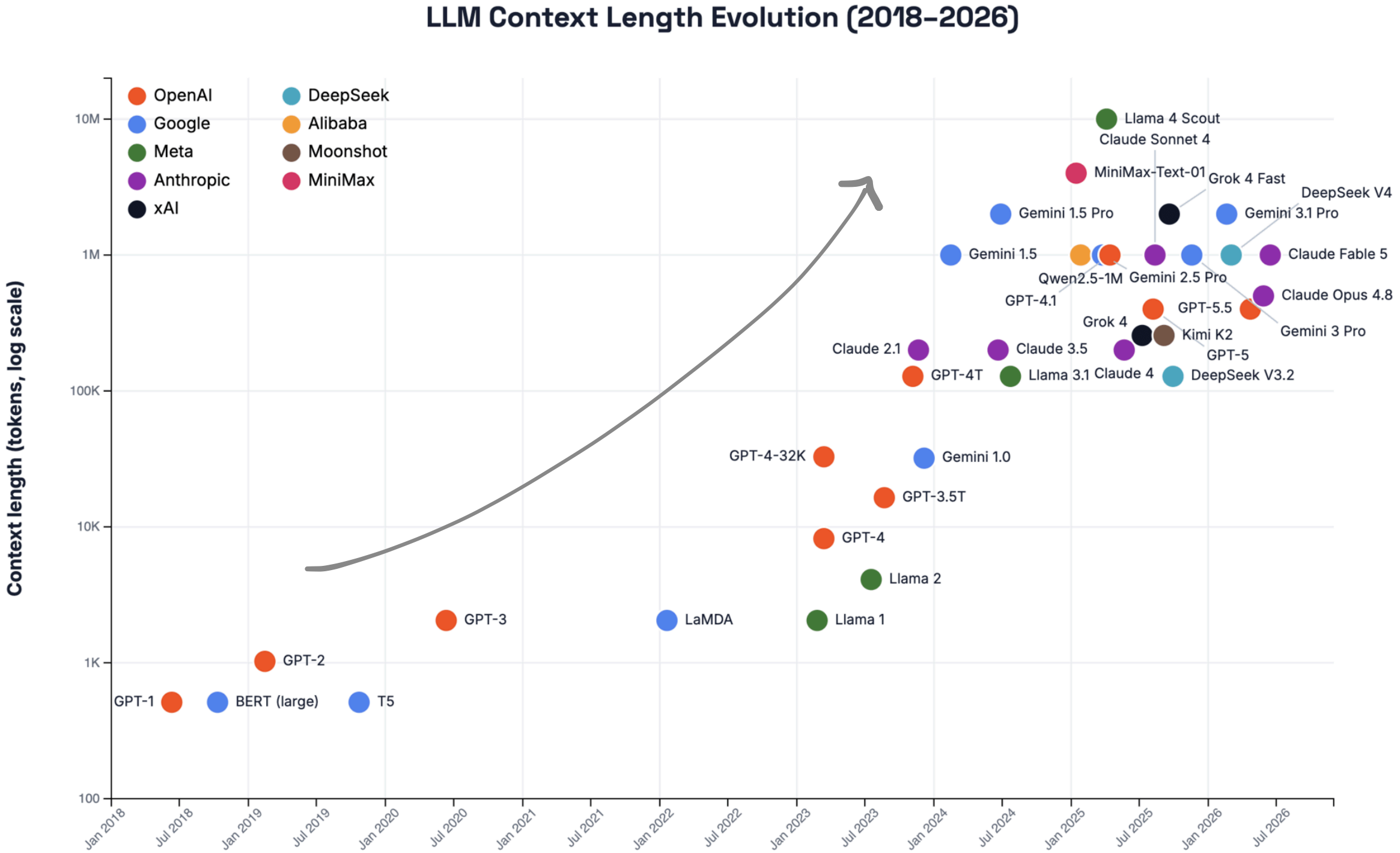
🚨 Not used in practice...

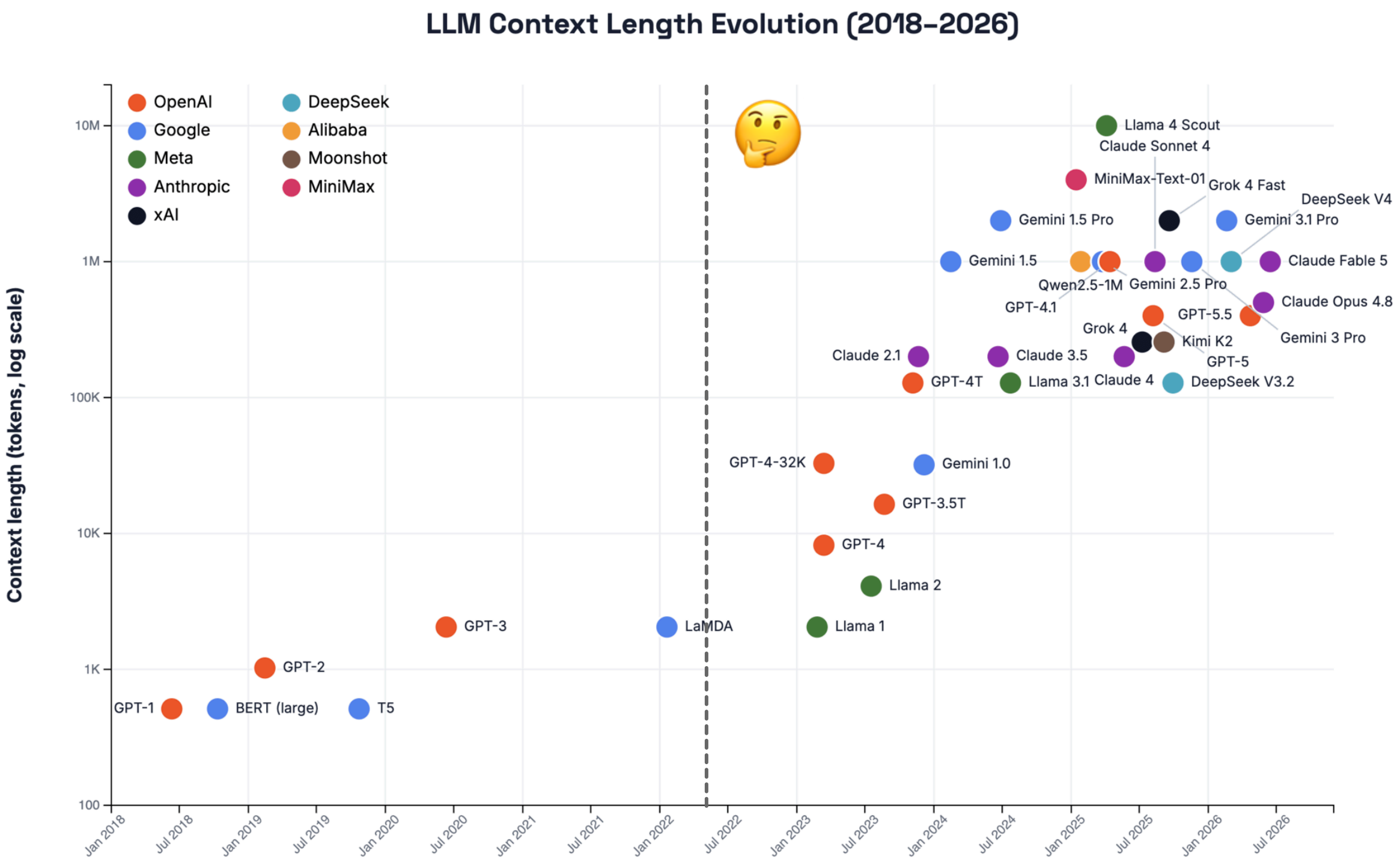
Why?

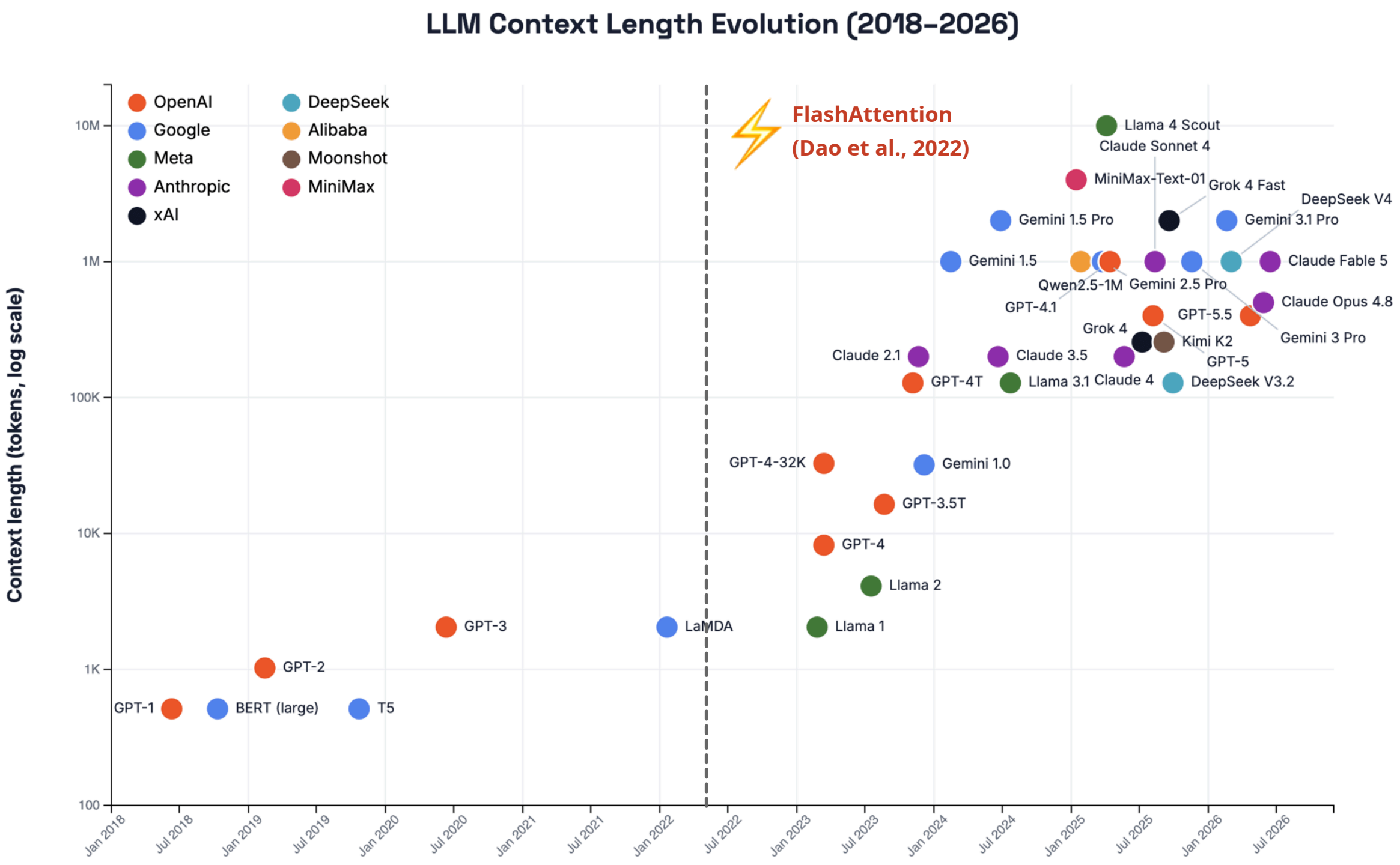
Accuracy was traded for efficiency!

Efficient Transformers: A Survey. Tay et al. (2020)


LLM Context Length Evolution (2018–2026)
Context length (tokens, log scale)
OpenAI
Google
Meta
Anthropic
xAI
DeepSeek
Alibaba
Moonshot
MiniMax
10M
1M
100K
10K
1K
100
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Jan 2025
Jul 2025
Jan 2026
Jul 2026
GPT-1
BERT (large)
T5
GPT-2
GPT-3
LaMDA
Llama 1
GPT-4-32K
Gemini 1.0
GPT-3.5T
GPT-4
Llama 2
Claude 2.1
GPT-4T
Claude 3.5
Llama 3.1
Llama 4 Scout
Claude Sonnet 4
Claude 4
MiniMax-Text-01
Grok 4 Fast
DeepSeek V4
Gemini 1.5 Pro
Gemini 3.1 Pro
Gemini 1.5
Qwen2.5-1M
Gemini 2.5 Pro
Claude Fable 5
GPT-4.1
Claude Opus 4.8
GPT-5.5
Grok 4
Kimi K2
Gemini 3 Pro
GPT-5
DeepSeek V3.2


LLM Context Length Evolution (2018–2026)
Context length (tokens, log scale)
OpenAI
Google
Meta
Anthropic
xAI
DeepSeek
Alibaba
Moonshot
MiniMax
10M
1M
100K
10K
1K
100
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Jan 2025
Jul 2025
Jan 2026
Jul 2026
GPT-1
BERT (large)
T5
GPT-2
GPT-3
LaMDA
Llama 1
GPT-4-32K
Gemini 1.0
GPT-3.5T
GPT-4
Llama 2
Claude 2.1
GPT-4T
Claude 3.5
Llama 3.1
Llama 4 Scout
Claude Sonnet 4
Claude 4
MiniMax-Text-01
Grok 4 Fast
DeepSeek V4
DeepSeek V3.2
Gemini 1.5 Pro
Gemini 3.1 Pro
Gemini 1.5
Claude Fable 5
Qwen2.5-1M
Gemini 2.5 Pro
GPT-4.1
Claude Opus 4.8
GPT-5.5
Grok 4
Kimi K2
Gemini 3 Pro
GPT-5


LLM Context Length Evolution (2018–2026)
Context length (tokens, log scale)
OpenAI
Google
Meta
Anthropic
xAI
DeepSeek
Alibaba
Moonshot
MiniMax
10M
1M
100K
10K
1K
100
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Jan 2025
Jul 2025
Jan 2026
Jul 2026
GPT-1
BERT (large)
T5
GPT-2
GPT-3
LaMDA
GPT-4-32K
Gemini 1.0
GPT-3.5T
GPT-4
Llama 2
Llama 1
Claude 2.1
GPT-4T
Claude 3.5
Llama 3.1
Llama 4 Scout
Claude Sonnet 4
Claude 4
MiniMax-Text-01
Grok 4 Fast
DeepSeek V4
Gemini 1.5 Pro
Gemini 3.1 Pro
Gemini 1.5
Qwen2.5-1M
Gemini 2.5 Pro
Claude Fable 5
GPT-4.1
Claude Opus 4.8
GPT-5.5
Grok 4
Kimi K2
Gemini 3 Pro
GPT-5
DeepSeek V3.2


LLM Context Length Evolution (2018–2026)
Context length (tokens, log scale)
OpenAI
Google
Meta
Anthropic
xAI
DeepSeek
Alibaba
Moonshot
MiniMax
FlashAttention
(Dao et al., 2022)
10M
1M
100K
10K
1K
100
Jan 2018
Jul 2018
Jan 2019
Jul 2019
Jan 2020
Jul 2020
Jan 2021
Jul 2021
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 2024
Jul 2024
Jan 2025
Jul 2025
Jan 2026
Jul 2026
GPT-1
BERT (large)
T5
GPT-2
GPT-3
LaMDA
Llama 1
Llama 2
GPT-4-32K
Gemini 1.0
GPT-3.5T
GPT-4
Claude 2.1
GPT-4T
Claude 3.5
Llama 3.1
Llama 4 Scout
Claude Sonnet 4
Claude 4
MiniMax-Text-01
Grok 4 Fast
DeepSeek V4
Gemini 1.5 Pro
Gemini 3.1 Pro
Gemini 1.5
Claude Fable 5
Qwen2.5-1M
Gemini 2.5 Pro
GPT-4.1
Claude Opus 4.8
GPT-5.5
Grok 4
Kimi K2
Gemini 3 Pro
GPT-5
DeepSeek V3.2

Where is the bottleneck?

```
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?

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



Z has $n \times n$ entries, materialized into memory

Where is the bottleneck?

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



Z has $n \times n$ entries, materialized into memory

```
1 O = torch.empty(n, d)
2 for i in range(n):
3   z = Q[i] @ K.T
4   p = torch.softmax(z, dim=-1)
5   O[i] = p @ V
```

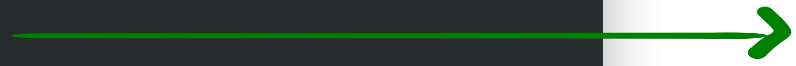
Where is the bottleneck?

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



Z has $n \times n$ entries, materialized into memory

```
1 O = torch.empty(n, d)
2 for i in range(n):
3   z = Q[i] @ K.T
4   p = torch.softmax(z, dim=-1)
5   O[i] = p @ V
```



only n materialized into memory

Where is the bottleneck?

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

Z has $n \times n$ entries, materialized into memory

```
1 O = torch.empty(n, d)
2 for i in range(n):
3   z = Q[i] @ K.T
4   p = torch.softmax(z, dim=-1)
5   O[i] = p @ V
```

for-loop is not parallel

only n materialized into memory

Where is the bottleneck?

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

Z has $n \times n$ entries, materialized into memory

```
1 O = torch.empty(n, d)
2 for i in range(n):
3   z = Q[i] @ K.T
4   p = torch.softmax(z, dim=-1)
5   O[i] = p @ V
```

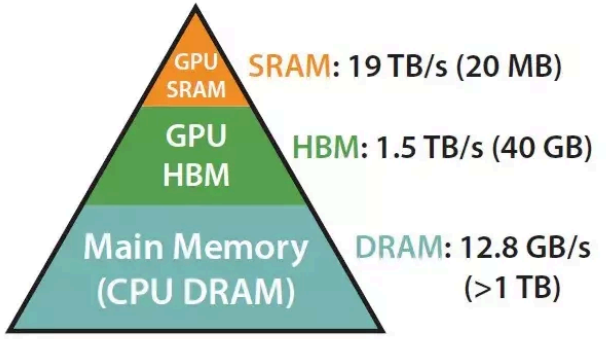
for-loop is not parallel

only n materialized into memory

🤔 So what exactly Flash Attention does?

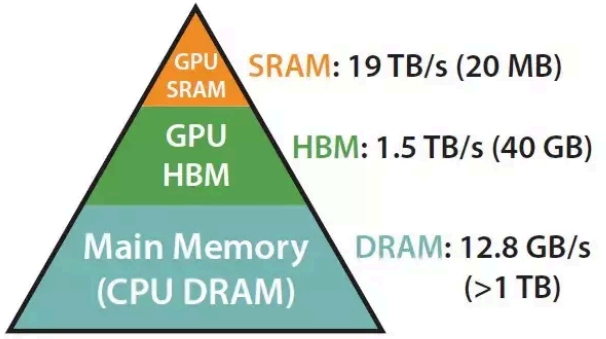
⚡Flash Attention

Hardware-aware implementation of softmax attention



⚡Flash Attention

Hardware-aware implementation of softmax attention

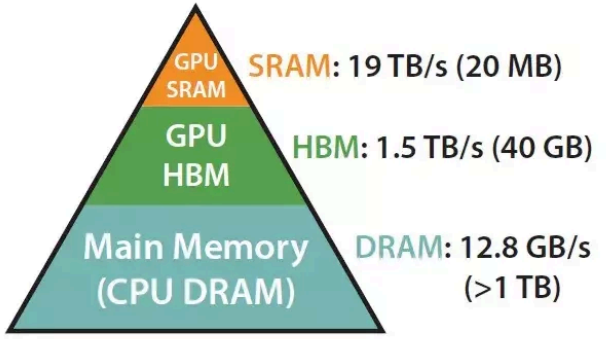



```
1 Z = torch.matmul(Q, K.transpose(-1, -2))
2 _
```

1. Load *Q* and *K* from **HBM** → compute *matmul* in **SRAM** → store *Z* in **HBM**

⚡Flash Attention

Hardware-aware implementation of softmax attention

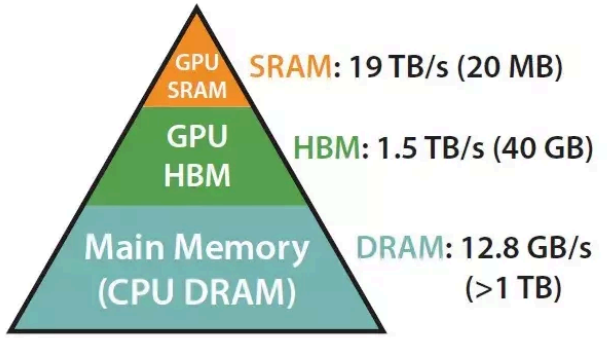



```
Z = torch.matmul(Q, K.transpose(-1, -2))
P = torch.softmax(Z, dim=-1)
_
```

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**

⚡Flash Attention

Hardware-aware implementation of softmax attention

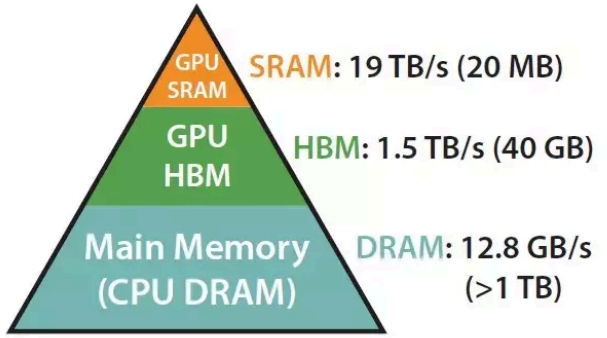



```
Z = torch.matmul(Q, K.transpose(-1, -2))
P = torch.softmax(Z, dim=-1)
O = torch.matmul(P, V)
```

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




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

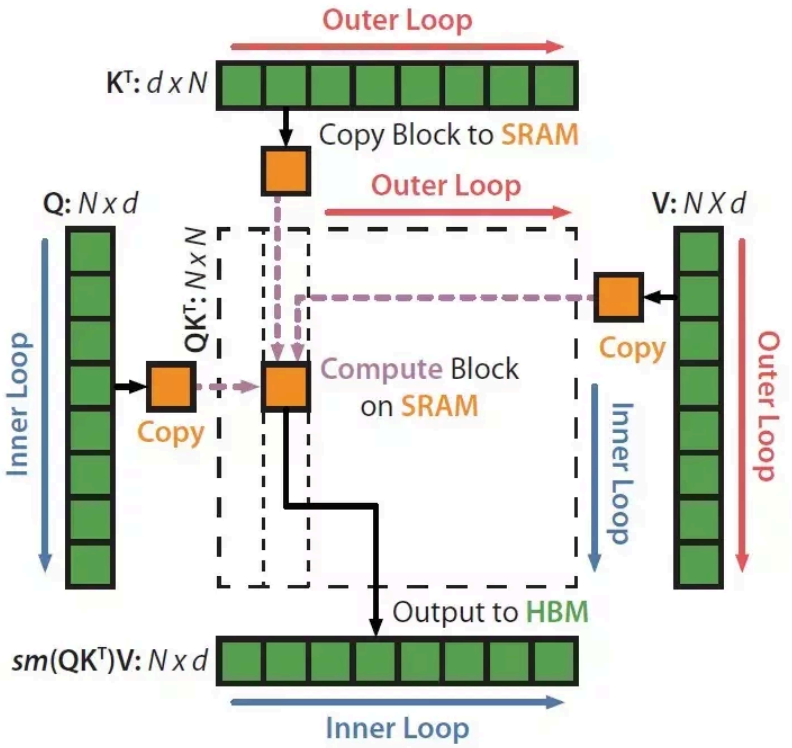
1. Load *Q* and *K* from **HBM** → compute *matmul* in **SRAM** → store *Z* in **HBM**
2. Load *Z* from **HBM** → compute *softmax* in **SRAM** → store *P* in **HBM**
3. Load *P* and *V* from **HBM** → compute matmul in **SRAM** → store *O* in **HBM**

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

Load *Q*, *K* and *V* from **HBM** → *compute Z* and *P* in **SRAM** → store *O* in **HBM**

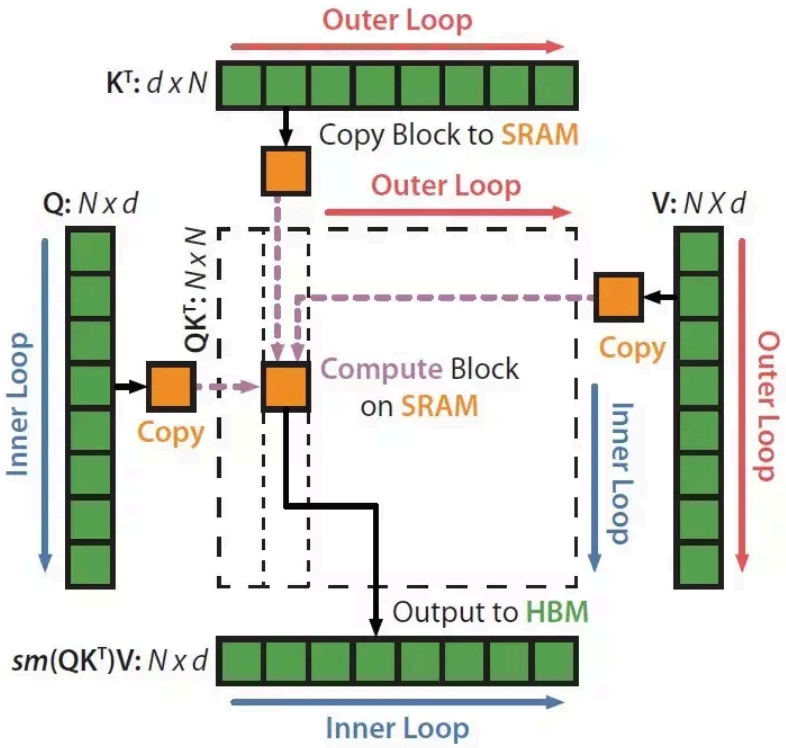
⚡Flash Attention

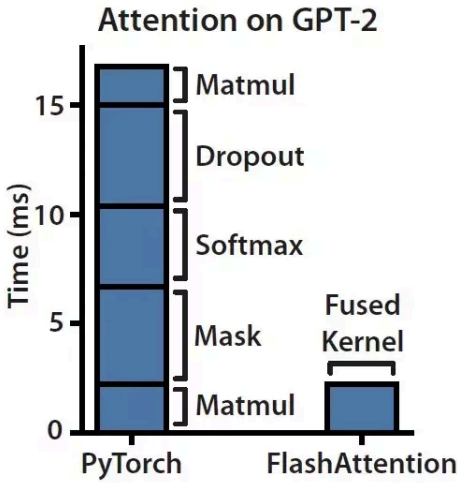
- **Tilling:** work with blocks rather than single vectors
- **Online softmax:** more FLOPs but more efficient while also being exact!
- **Recomputation:** how to backprop through this?
  $\rightarrow$ recompute in the backward pass

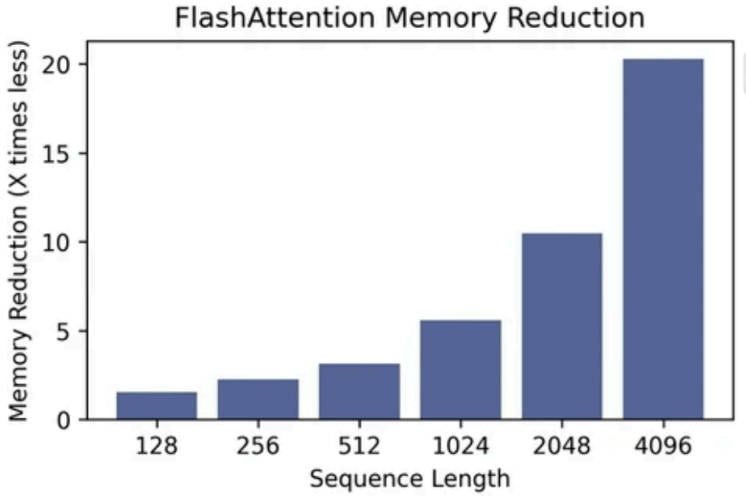


⚡Flash Attention

- **Tilling:** 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 in the backward pass



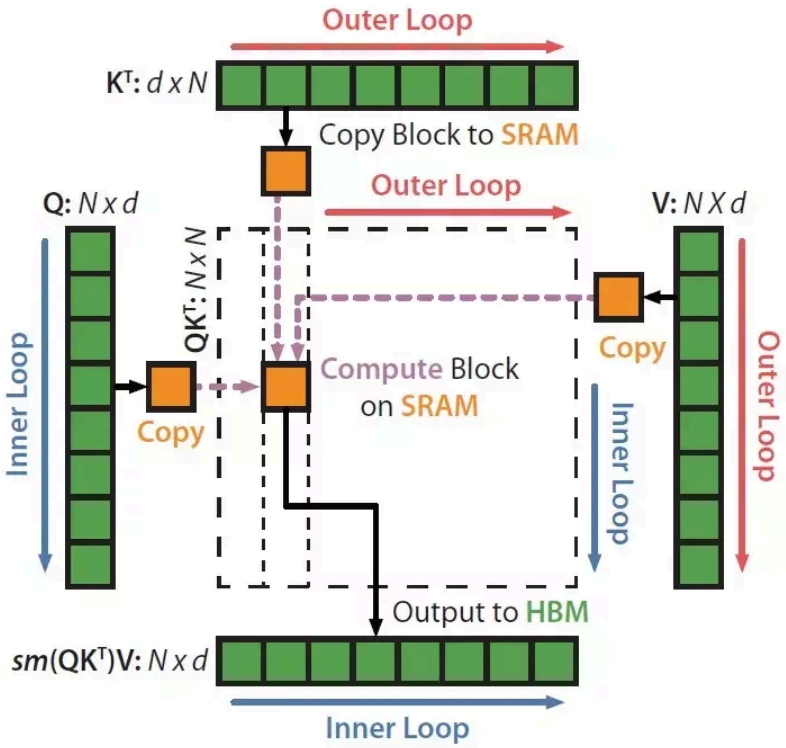


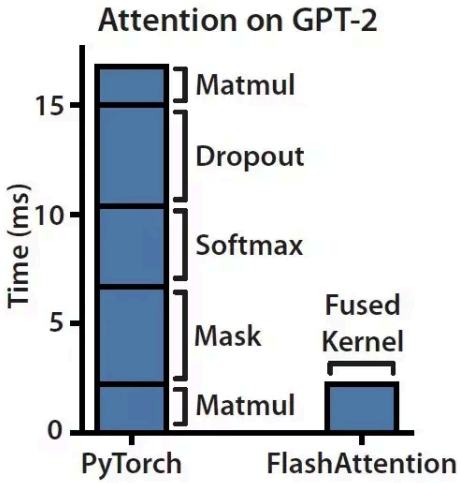


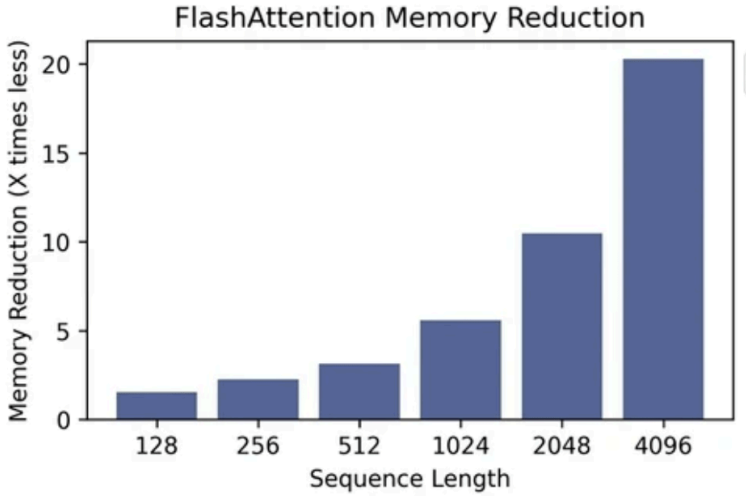
~4x faster
~20x less memory!

⚡Flash Attention

- **Tilling:** 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 in the backward pass







~4x faster
~20x less memory!

FA-2 = base algorithm
FA-3 → Hopper
FA-4 → Blackwell

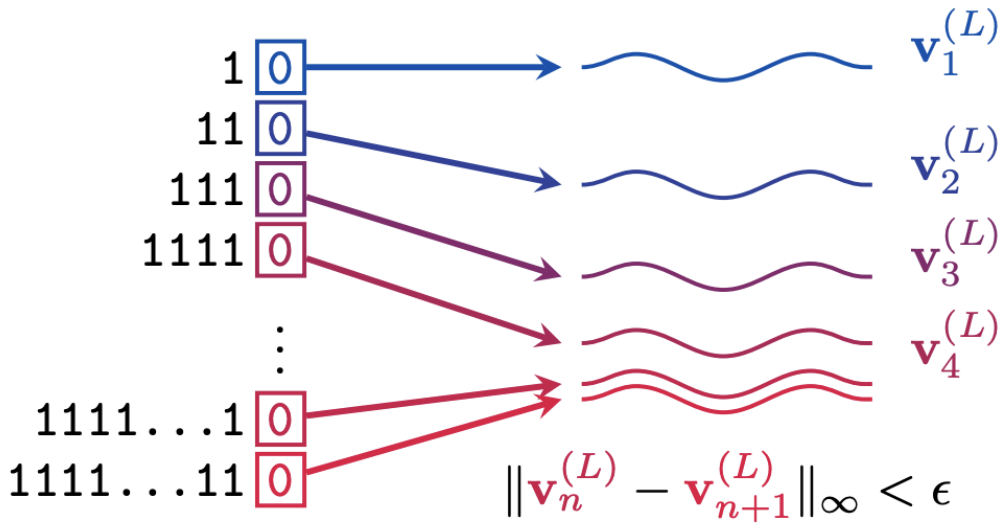
Back to Attention

- Can we just keep increasing the context length?

Back to Attention

- Can we just keep increasing the context length?

Representational Collapse

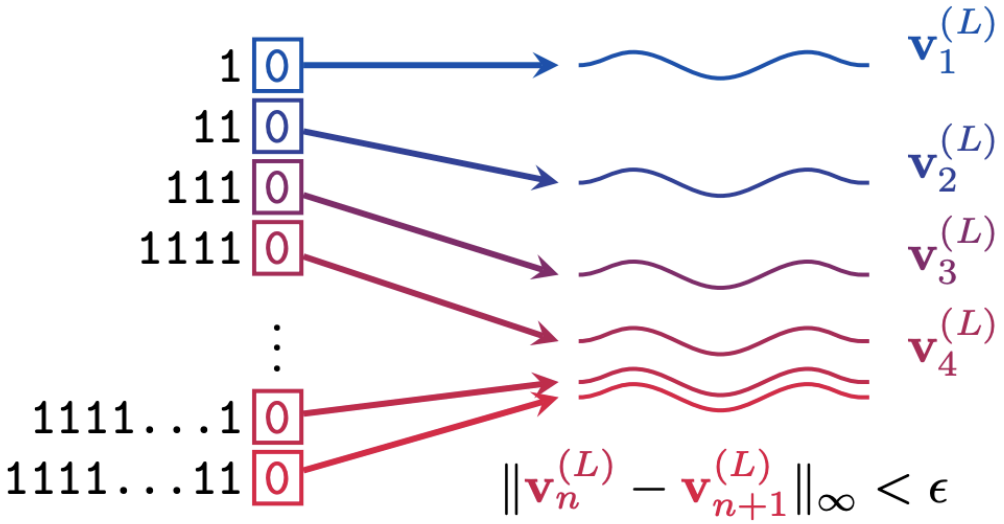


As we add context tokens, the final representation of "0" collapse to a single vector

Back to Attention

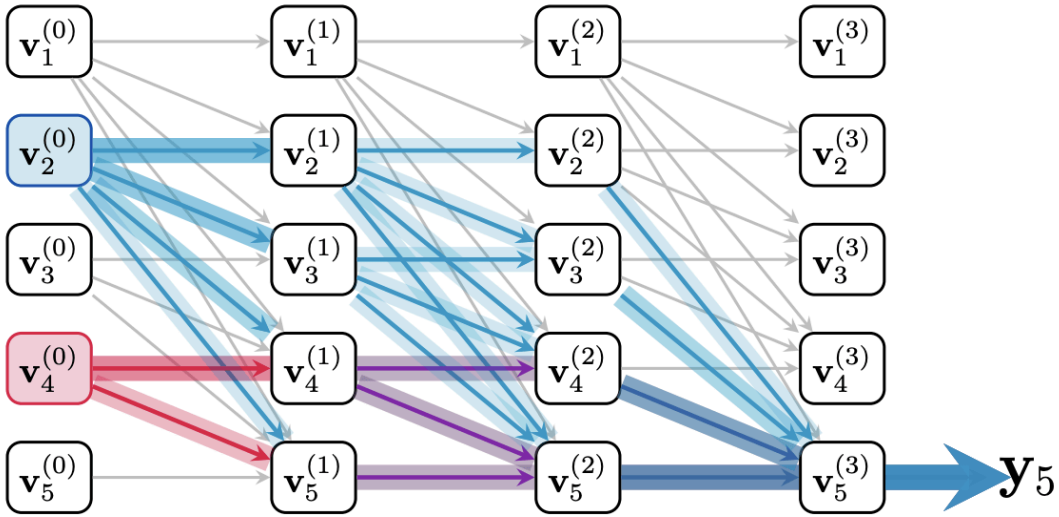
- Can we just keep increasing the context length?

Representational Collapse



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

Oversquashing

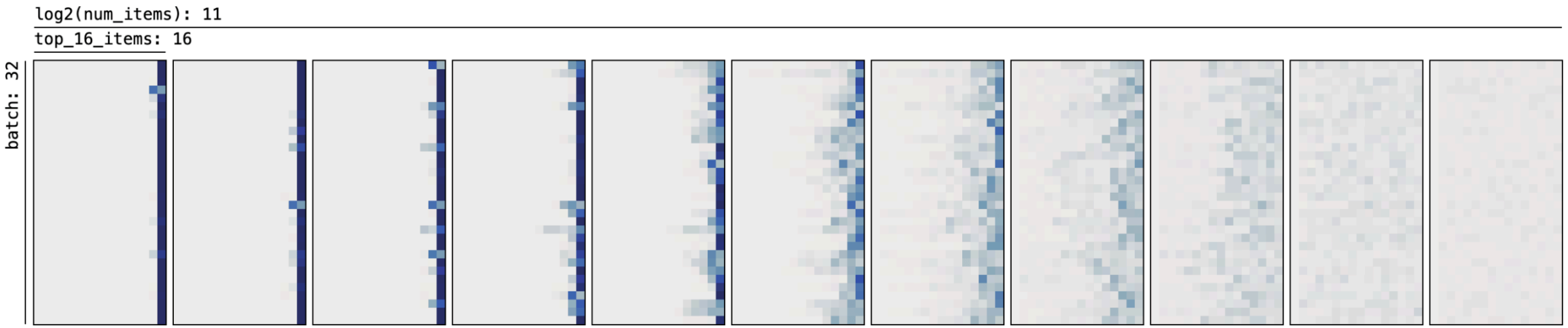


A lot of information from context tokens is forced through a single, fixed-size bottleneck

Back to Attention

- Can we just keep increasing the context length?

Attention Dispersion

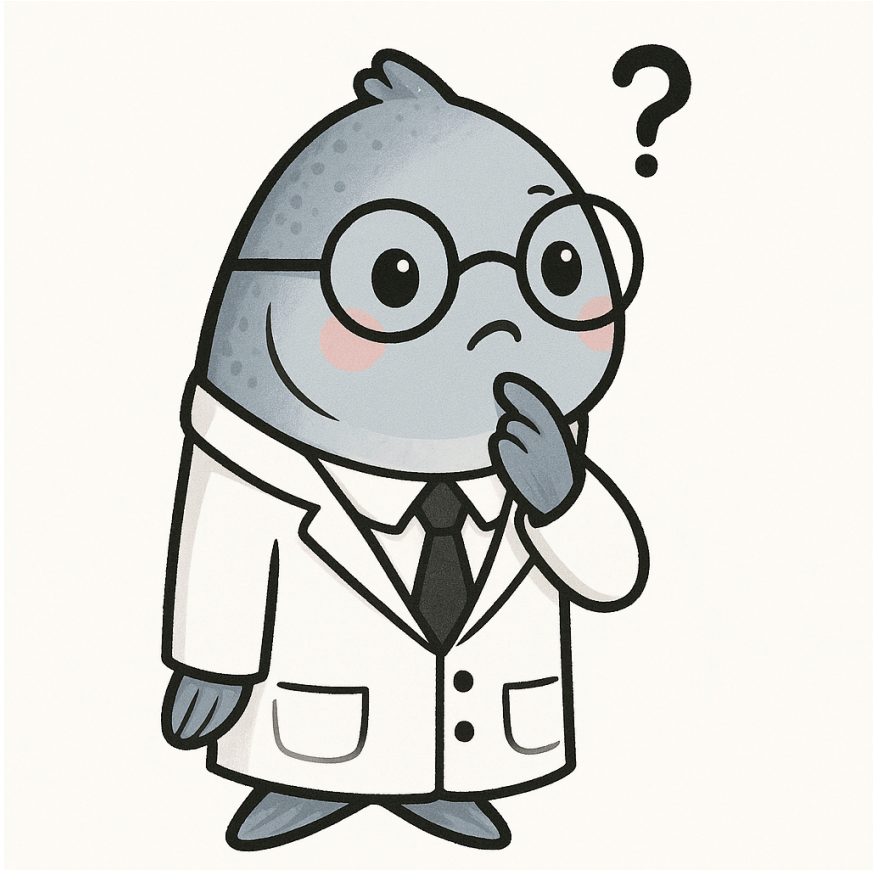


As the number of tokens increase, the attention coefficients spread out

→ uniform distribution (simple average)

Softmax is not Enough (for Sharp Size Generalisation). Velickovic et al., ICML 2025

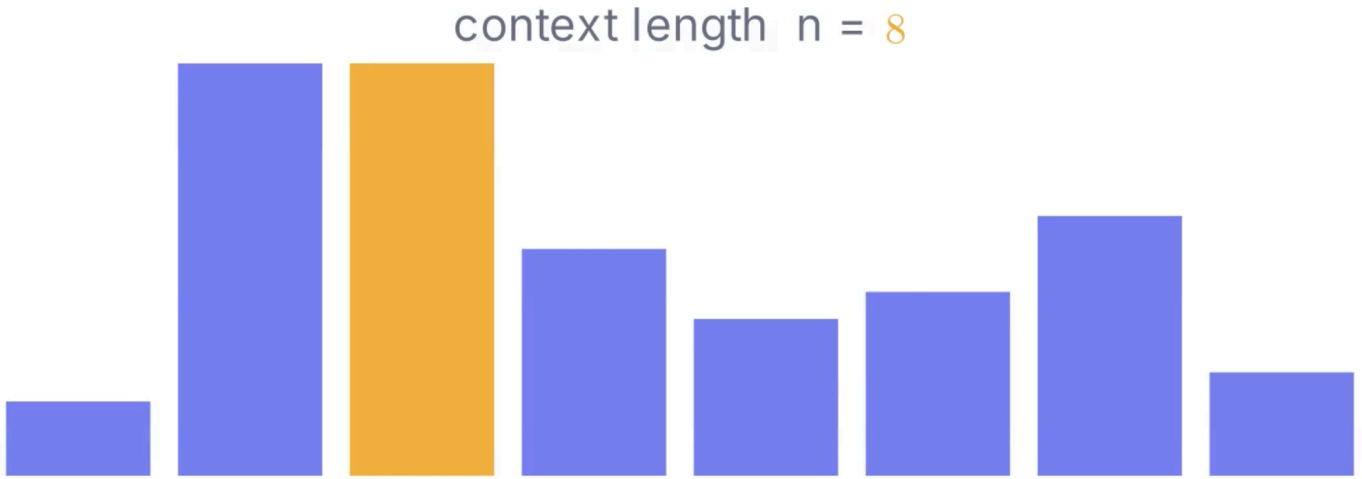
What is going on?



What is going on?

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

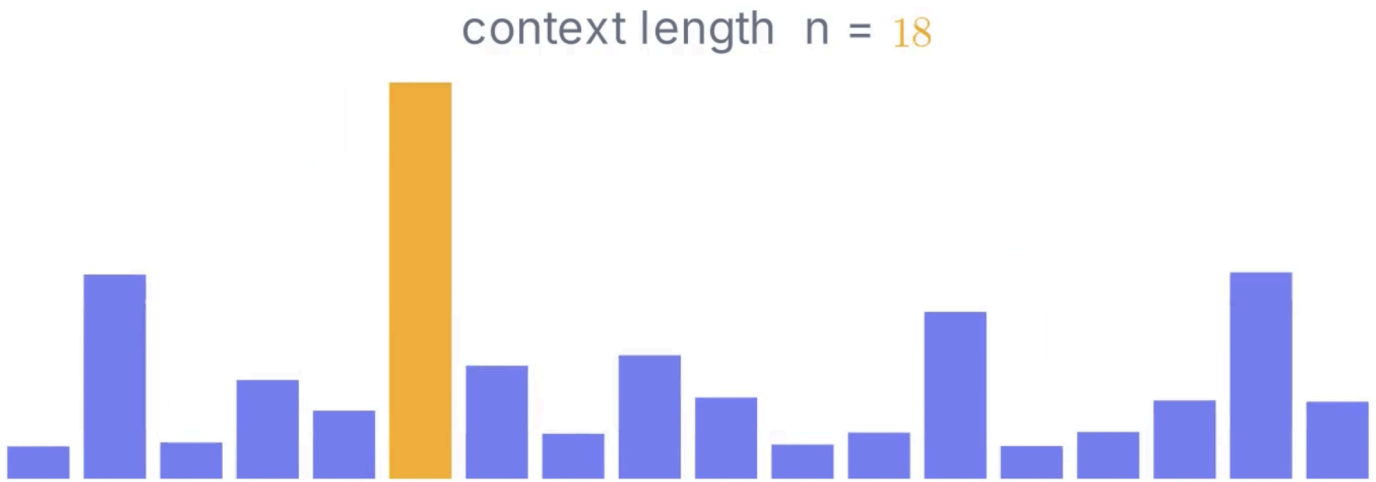
$$\text{softmax}(\boldsymbol{z})_j = \frac{\exp(\boldsymbol{z}_j)}{\sum_{k=1}^{n} \exp(\boldsymbol{z}_k)}$$



What is going on?

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

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



What is going on?

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

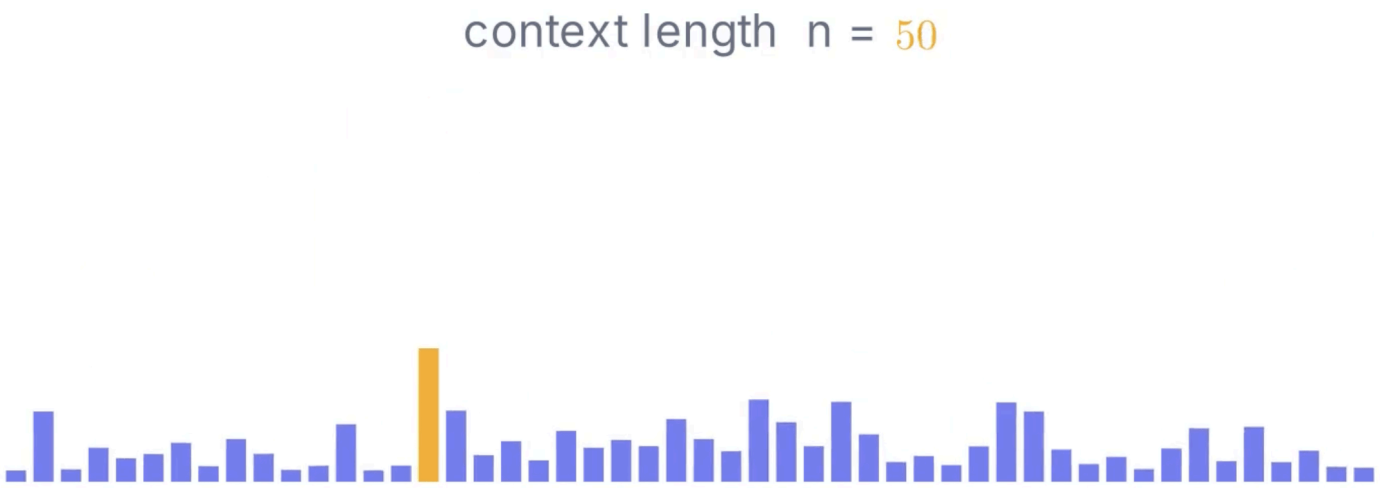
$$\text{softmax}(\boldsymbol{z})_j = \frac{\exp(\boldsymbol{z}_j)}{\sum_{k=1}^{n} \exp(\boldsymbol{z}_k)}$$



What is going on?

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

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



What is going on?

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

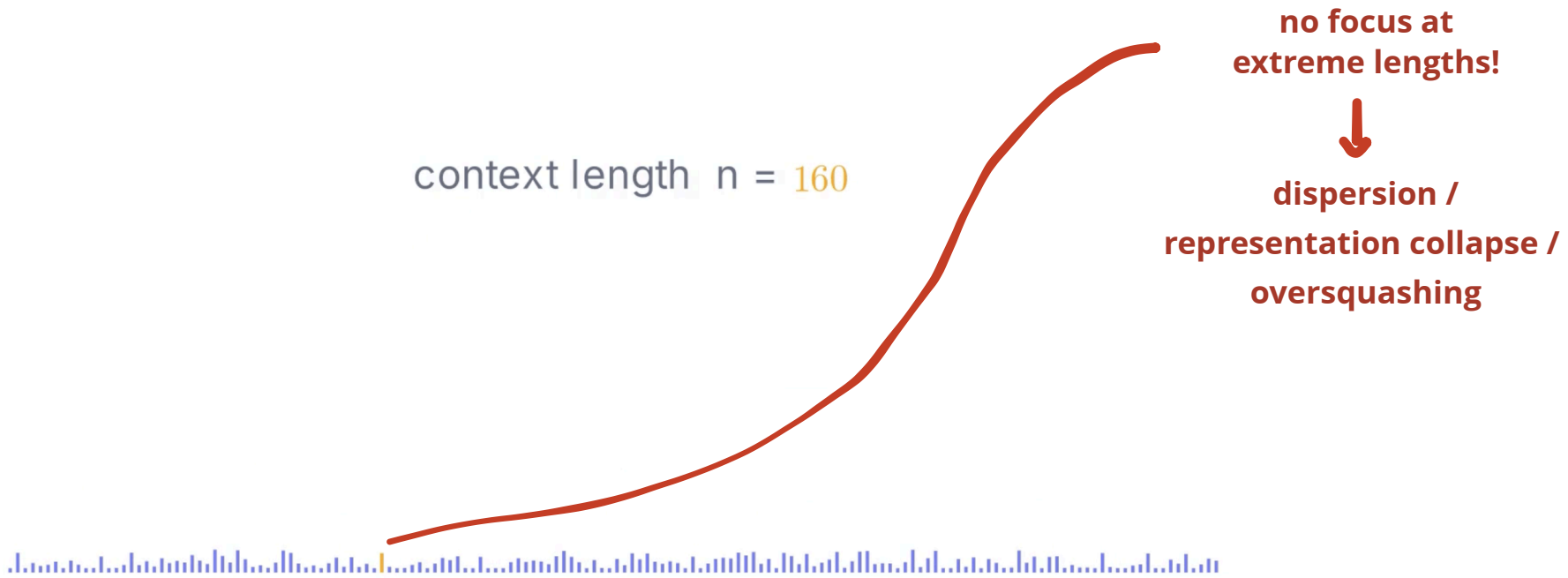
$$\text{softmax}(\boldsymbol{z})_j = \frac{\exp(\boldsymbol{z}_j)}{\sum_{k=1}^{n} \exp(\boldsymbol{z}_k)}$$



What is going on?

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

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

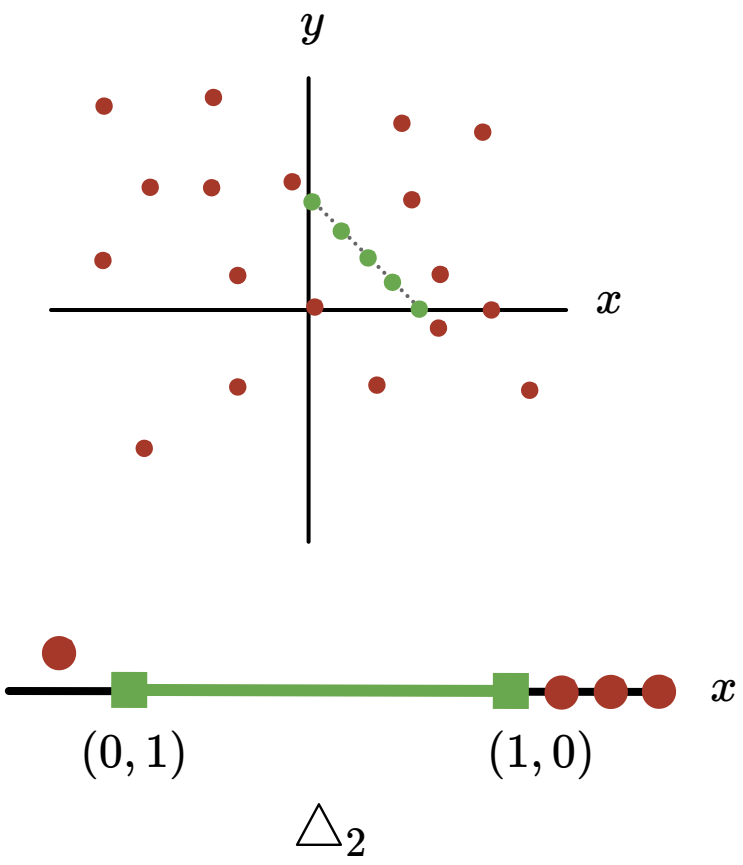


Softmax origins: probability simplex

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

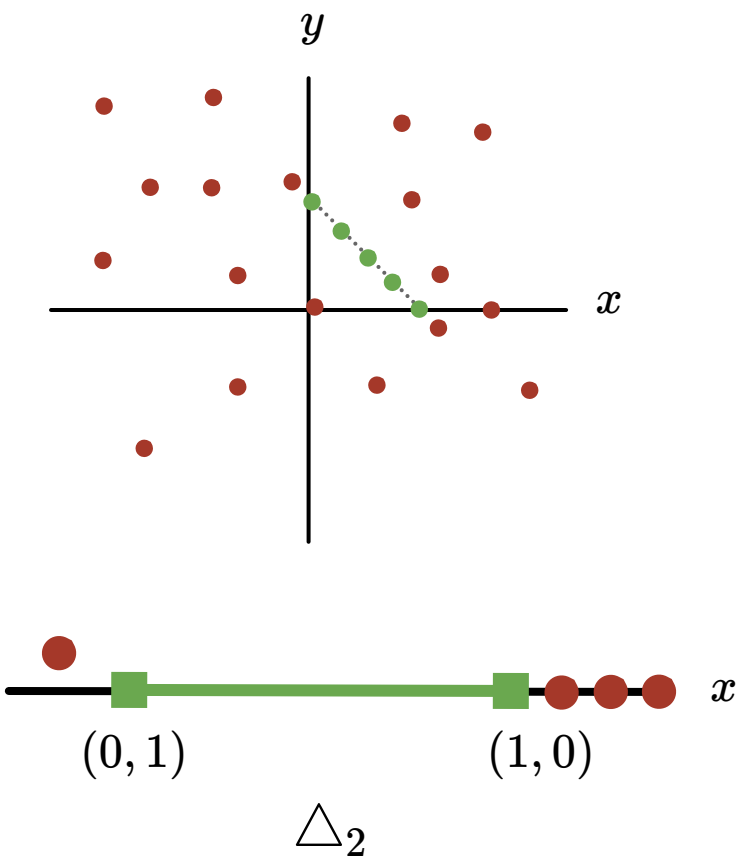
Softmax origins: probability simplex

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

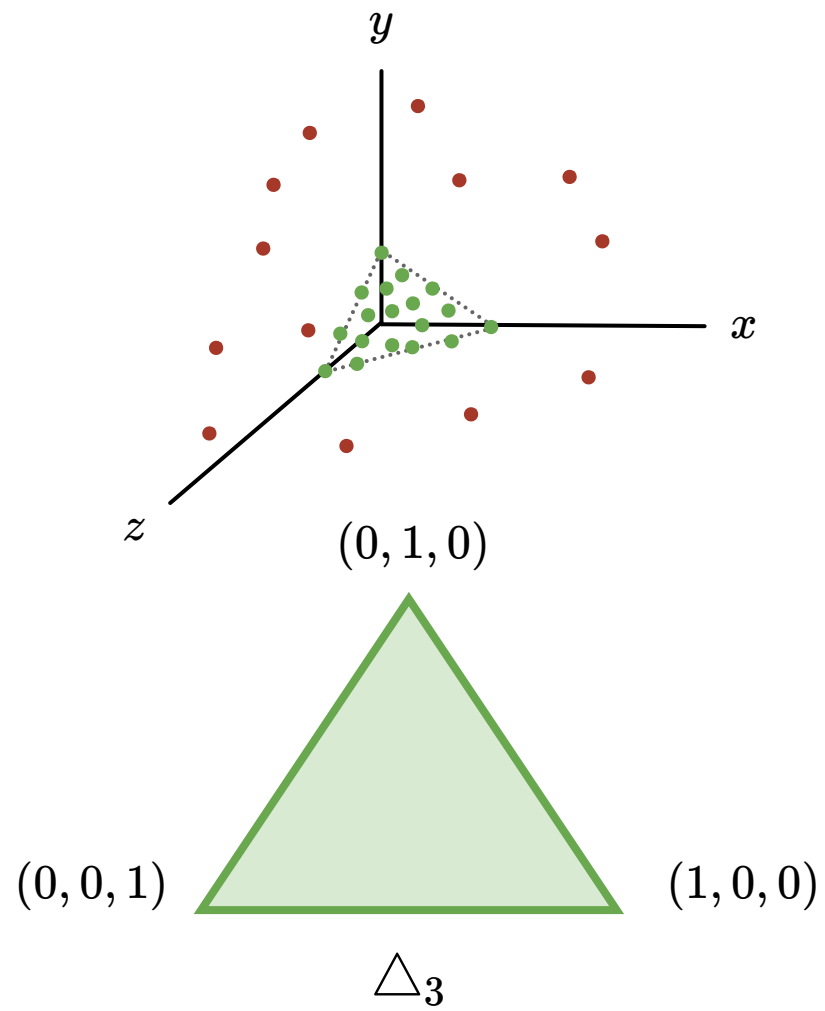


Softmax origins: probability simplex

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



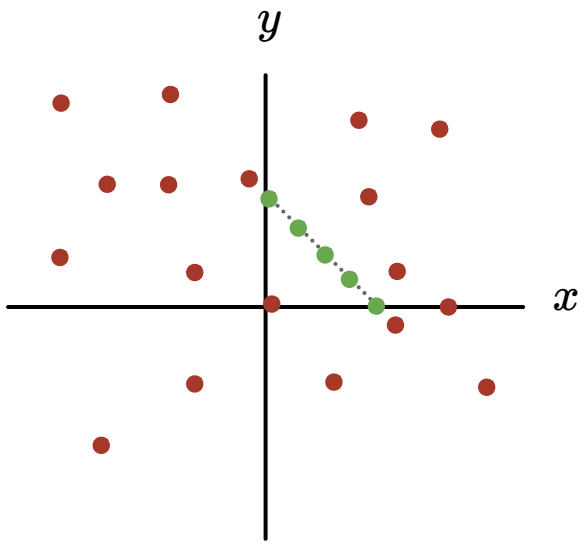
$\triangle_2$

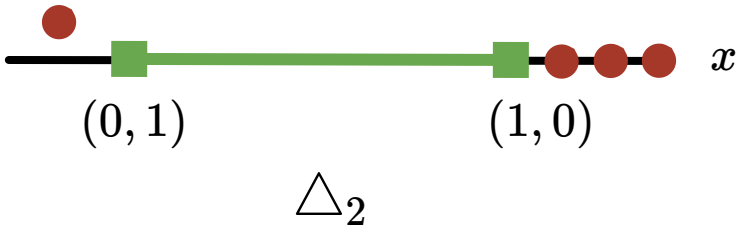


$\triangle_3$

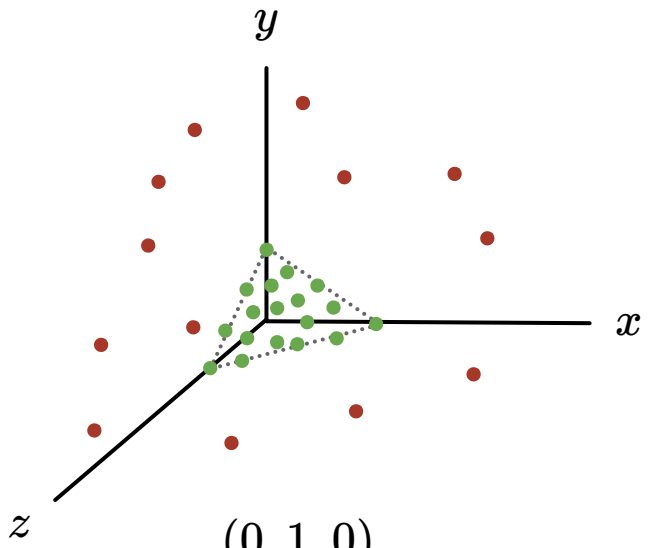
Softmax origins: probability simplex

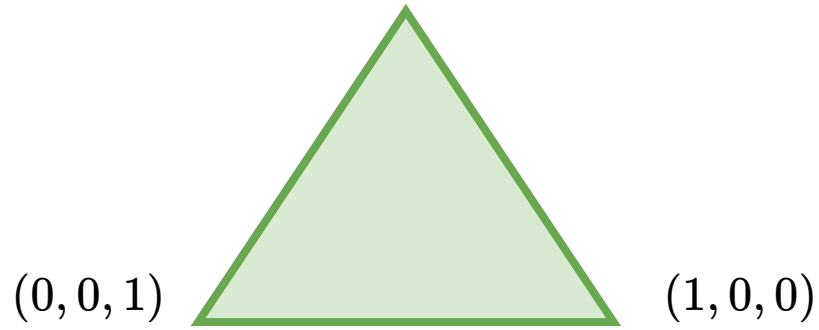
$$\triangle_n = \{\boldsymbol{p} \in \mathbb{R}^n_+ \mid \textstyle\sum_i p_i = 1\}$$





$\triangle_2$



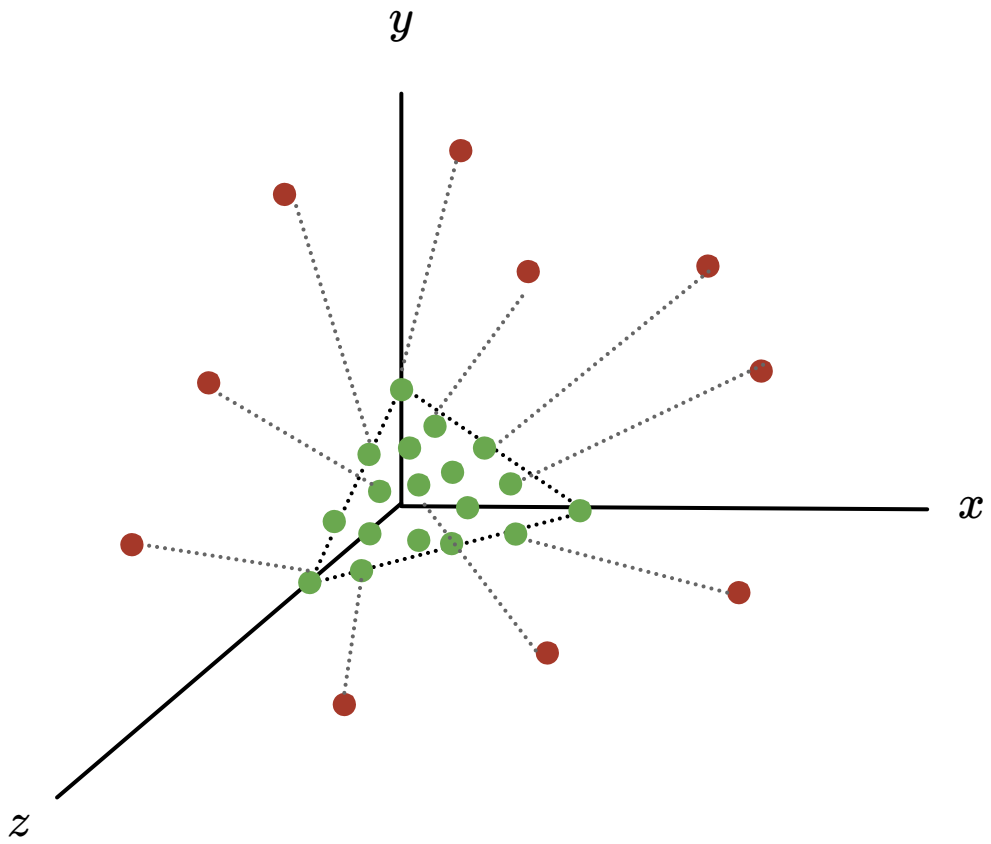


$\triangle_3$



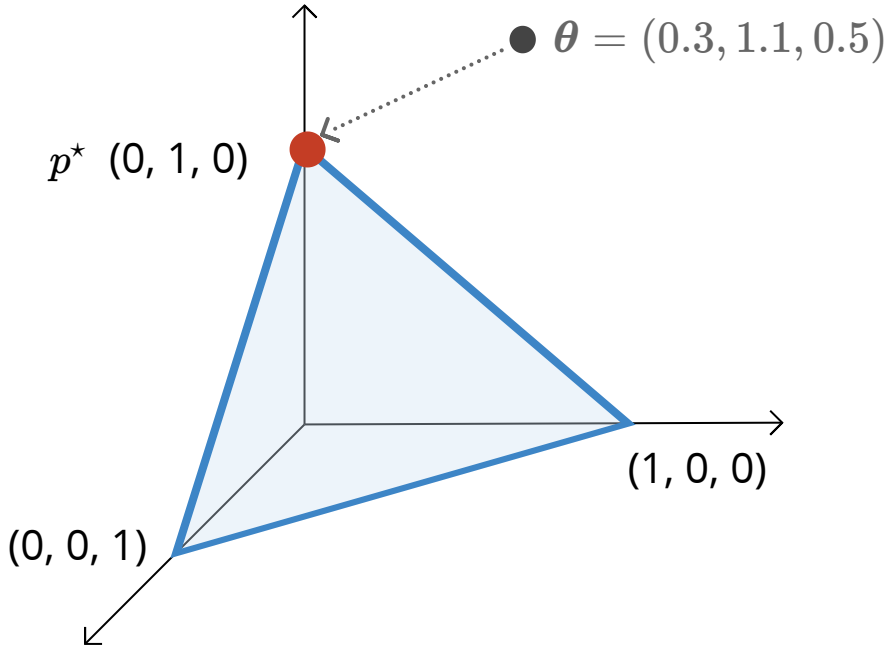
$\triangle_4$

How can we map a score to a probability?



The simplest option: argmax

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



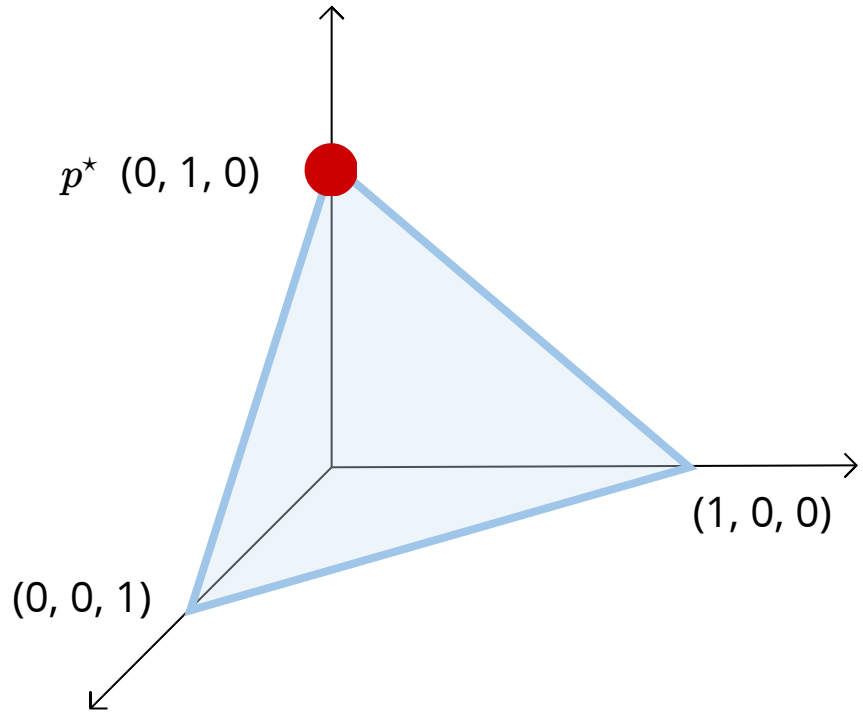
Fundamental Thm. Lin. Prog.
(Dantzig et al., 1955)

What if we regularize the opt. problem?

Entropy regularizer

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

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



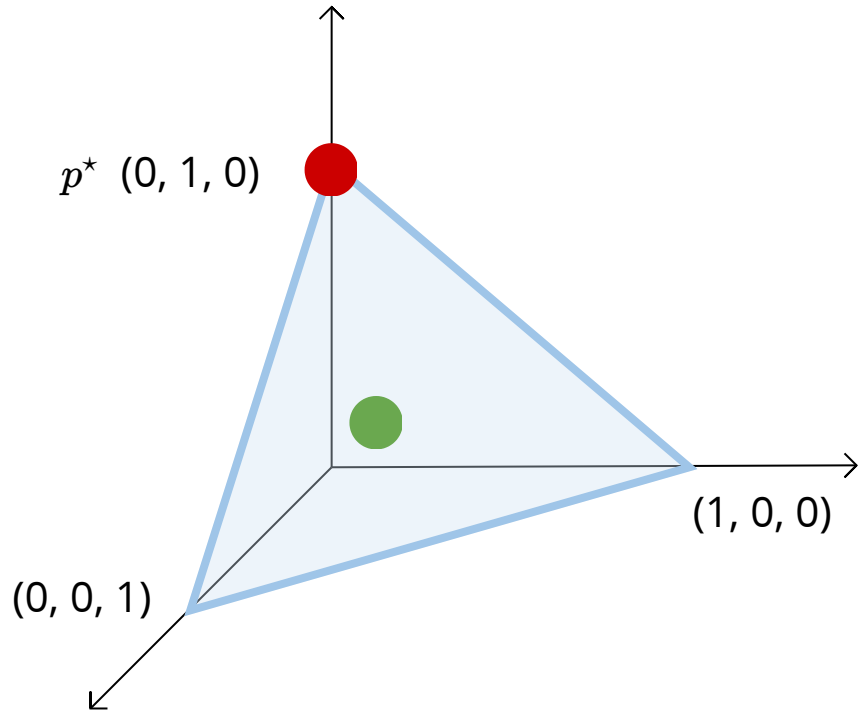
(Martins and Astudillo, 2016)
(Niculae and Blondel, 2017)
(Peters et al., 2019)

Softargmax is just a regularized version

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



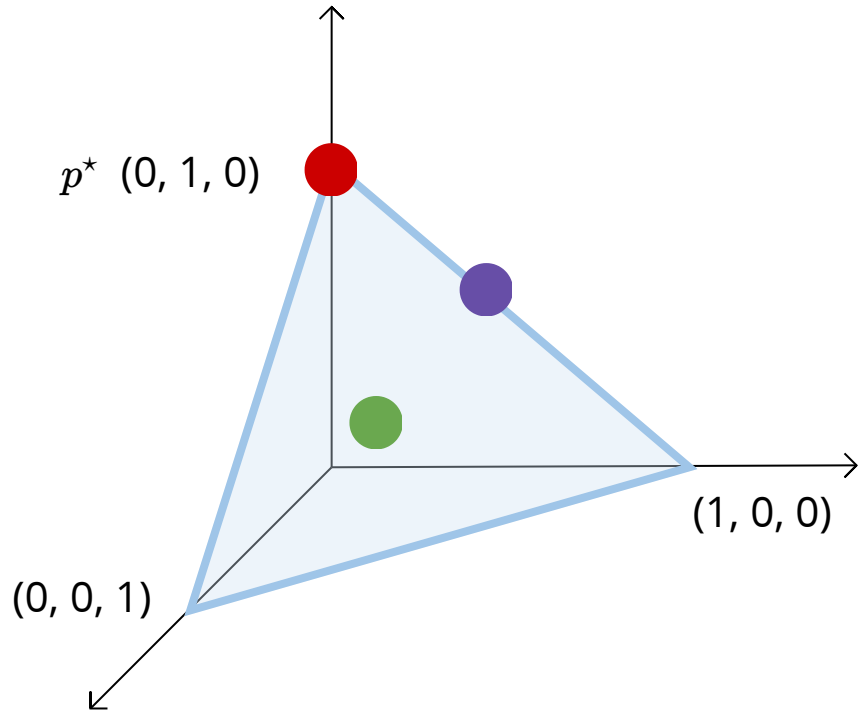
(Martins and Astudillo, 2016)
(Niculae and Blondel, 2017)
(Peters et al., 2019)

Sparsemax

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)
(Niculae and Blondel, 2017)
(Peters et al., 2019)

α-entmax

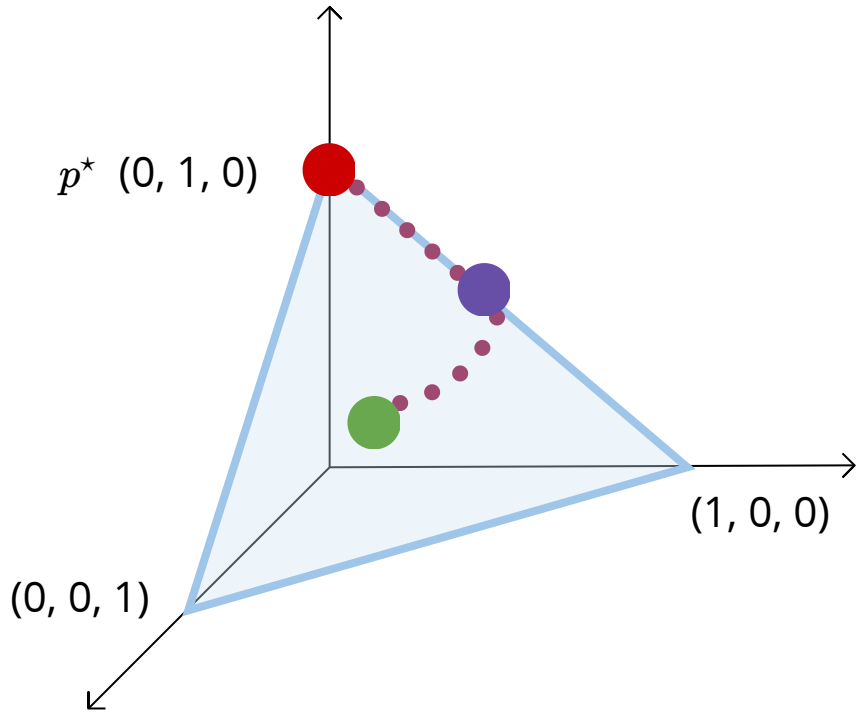
Entropy regularizer

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

- 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 $\alpha = 2$



(Martins and Astudillo, 2016)
(Niculae and Blondel, 2017)
(Peters et al., 2019)

α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

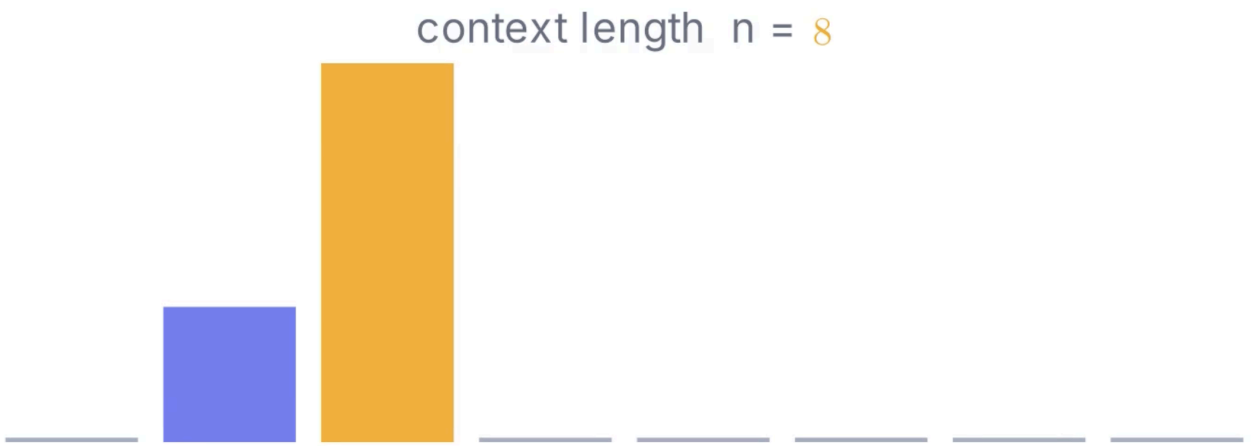
such that $\sum_i p_i = 1$

α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$

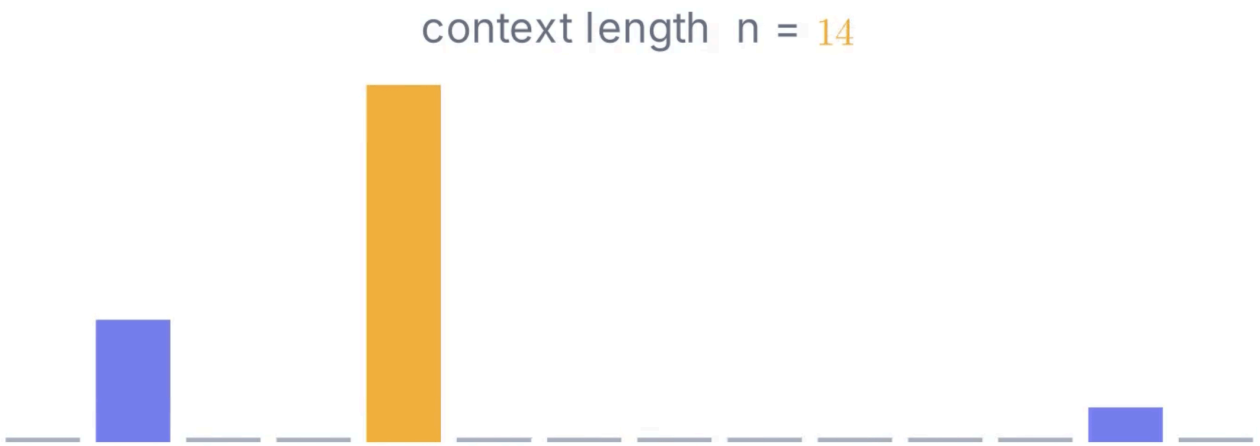


α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$

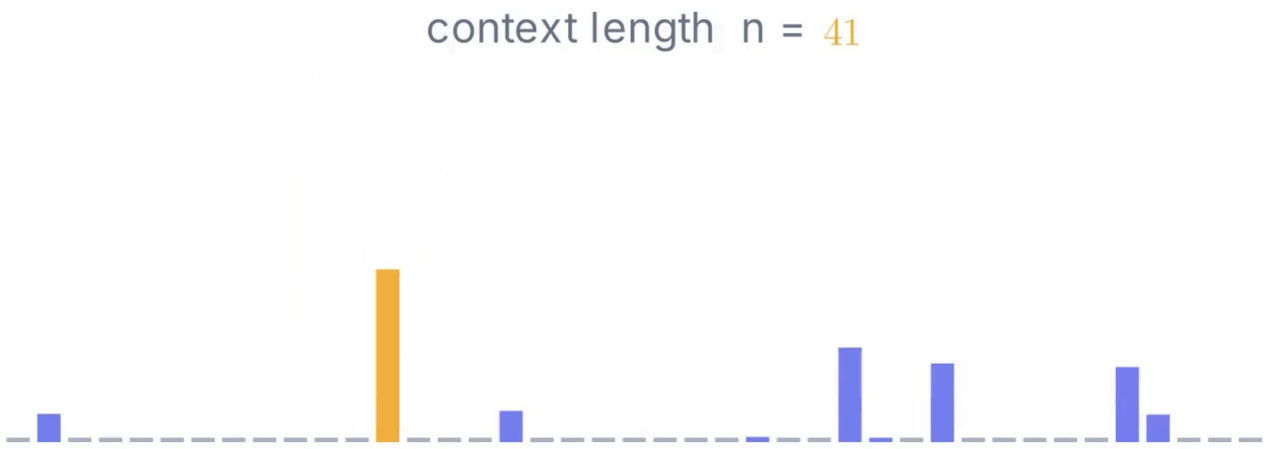


α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$

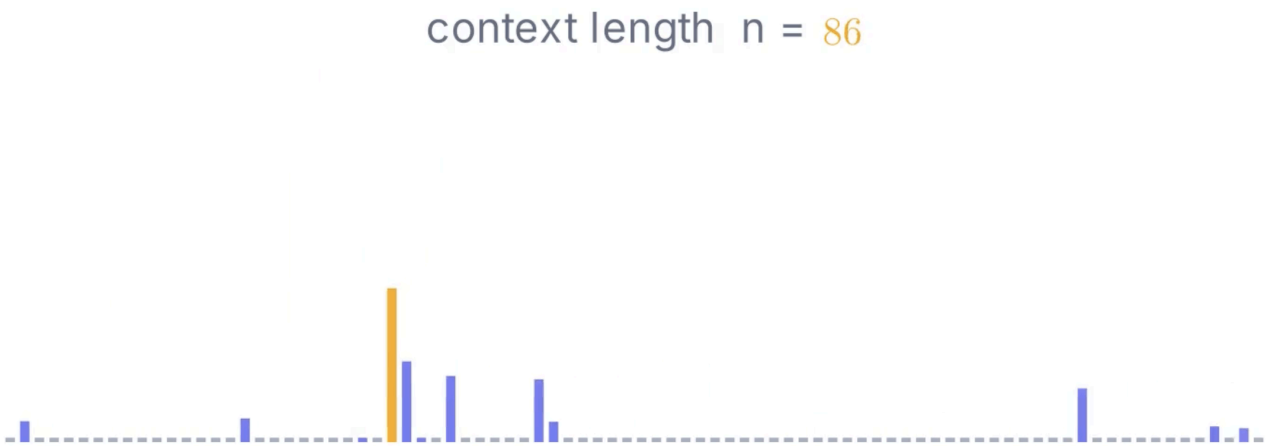


α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$

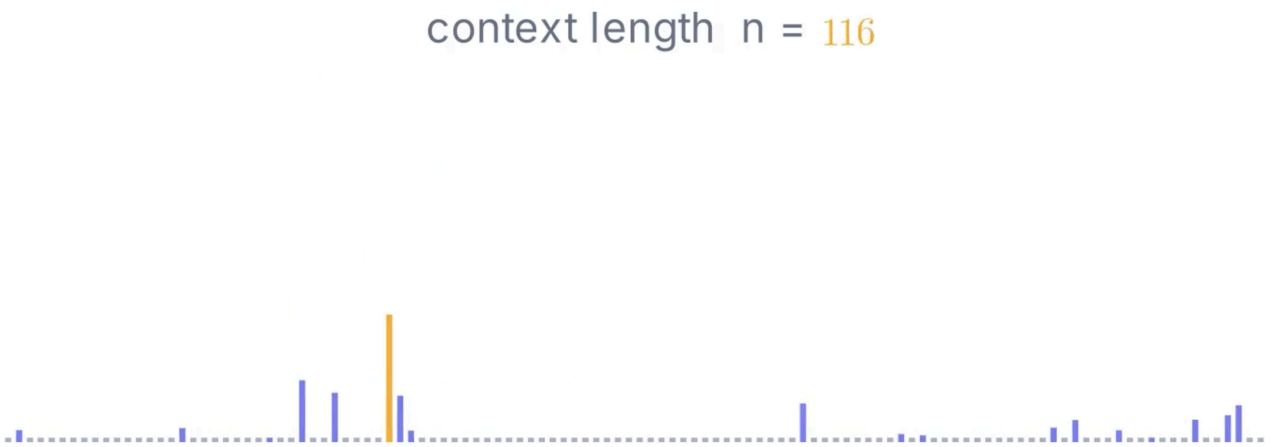


α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$

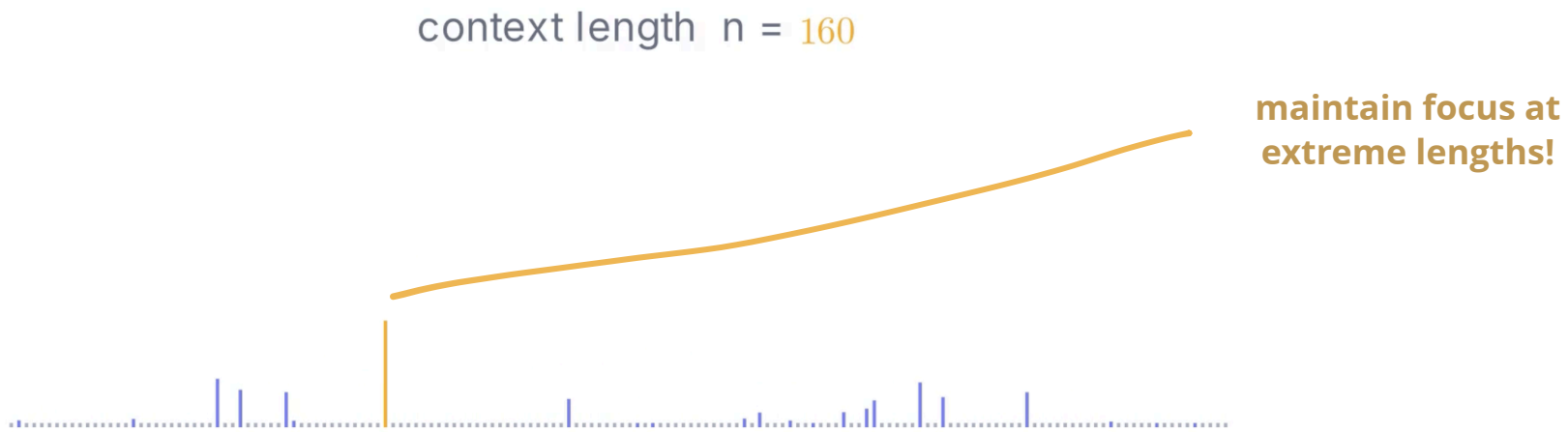


α-entmax

The output of $\alpha\text{-entmax}(\boldsymbol{z})$ is quite simple:

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

such that $\sum_i p_i = 1$



α-entmax

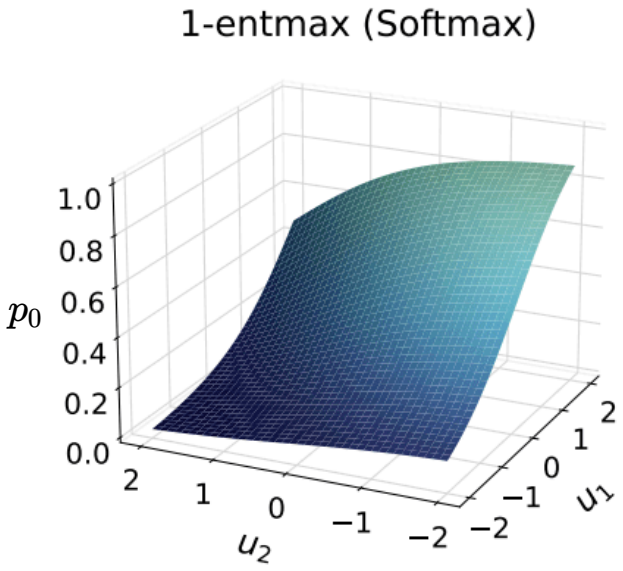


But, why complicate so much?

If we want sparsity, why just not use top-k?

$$\boldsymbol{u} = [0, u_1, u_2]$$

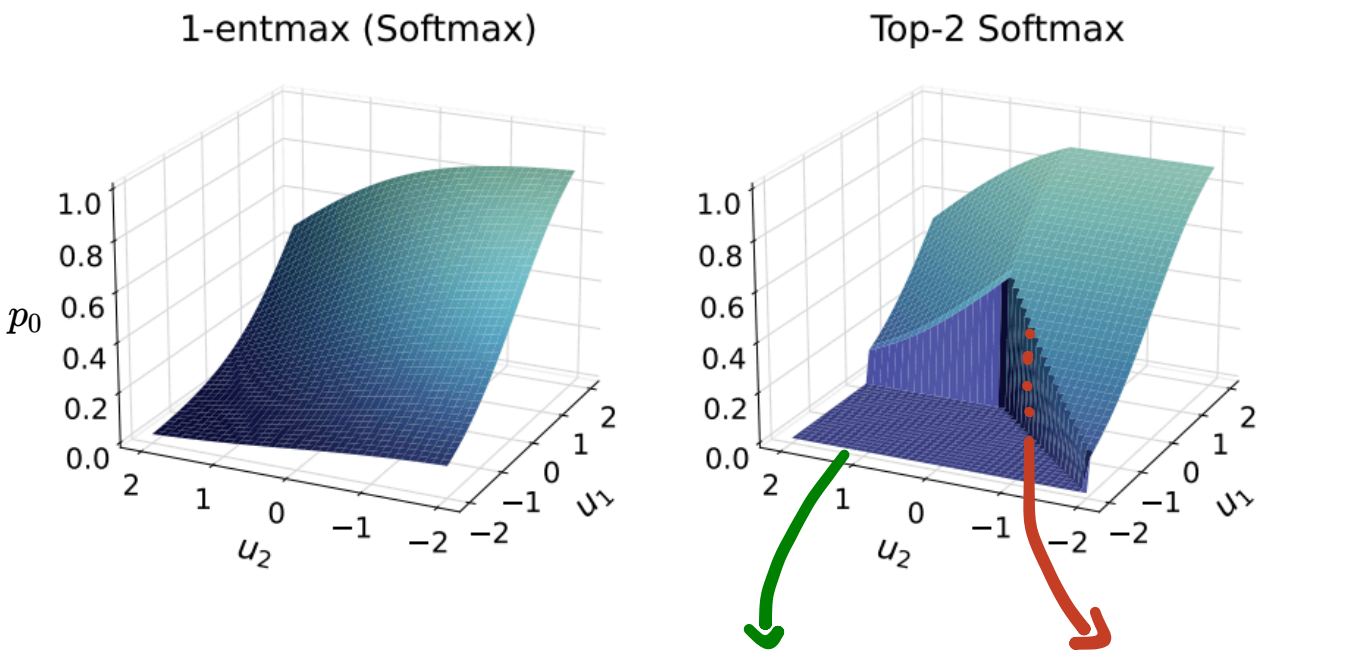
$$\boldsymbol{p} = f(\boldsymbol{u})$$



dense and differentiable

$$\boldsymbol{u} = [0, u_1, u_2]$$

$$\boldsymbol{p} = f(\boldsymbol{u})$$

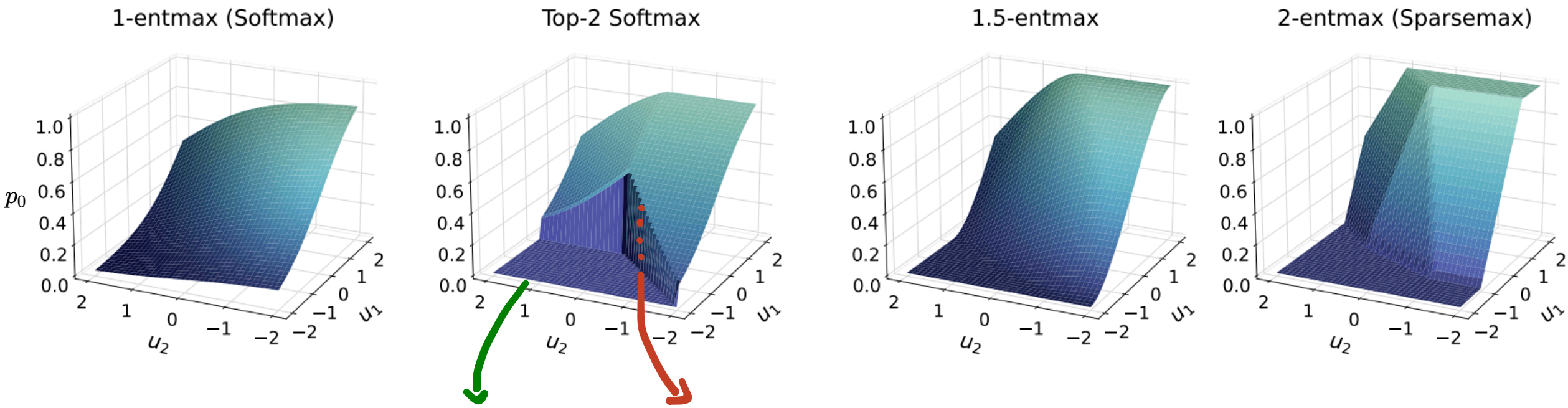


dense and differentiable

sparse and non-differentiable

$$\boldsymbol{u} = [0, u_1, u_2]$$

$$\boldsymbol{p} = f(\boldsymbol{u})$$



dense and differentiable

sparse and non-differentiable

sparse and differentiable

to be mathematically precise, α-entmax is *differentiable almost everywhere*

Entmax Solver

such that $\sum_i p_i = 1$

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

Entmax Solver

such that $\sum_i p_i = 1$

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

- $\alpha = 2$ (sparsemax)

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

(Martins and Astudillo, 2016)

- $\alpha = 1.5$

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**
  $M(\rho) \leftarrow 1/\rho \sum_{j=1}^{\rho} z_{[j]}$
  $S(\rho) \leftarrow \sum_{j=1}^{\rho} \left(z_{[j]} - M(\rho)\right)^2$
  $\tau(\rho) \leftarrow M(\rho) - \sqrt{1/\rho \left(1 - S(\rho)\right)}$
  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)

Entmax Solver

such that $\sum_i p_i = 1$

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

- $\alpha = 2$ (sparsemax)

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

(Martins and Astudillo, 2016)

- $\alpha = 1.5$

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**
$M(\rho) \leftarrow 1/\rho \sum_{j=1}^{\rho} z_{[j]}$
$S(\rho) \leftarrow \sum_{j=1}^{\rho} \left(z_{[j]} - M(\rho)\right)^2$
$\tau(\rho) \leftarrow M(\rho) - \sqrt{1/\rho \left(1 - S(\rho)\right)}$
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)

- 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

📌 ICML 2025 (spotlight)

📄 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)
```

Entmax Solver

$$f(\tau) := \sum_j p_j - 1$$

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

with $f(\tau^\star) = 0$

Entmax Solver

$$f(\tau) := \sum_j p_j - 1$$

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

with $f(\tau^\star) = 0$

- Solve for $f(\tau) = 0 \rightarrow$ root-finding!
- $f(\tau)$ is *strictly* convex for 1 < α ≤ 2 (unique root)
- One evaluation of $f(\tau)$ = one pass over K
- Bisection:

$$\underbrace{\max_j z_j - \frac{1}{\alpha-1}}_{\text{One-hot solution}} \leq \tau < \underbrace{\max_j z_j}_{\substack{\text{Fully dense case} \\ \text{And every score the same}}}$$

Entmax Solver

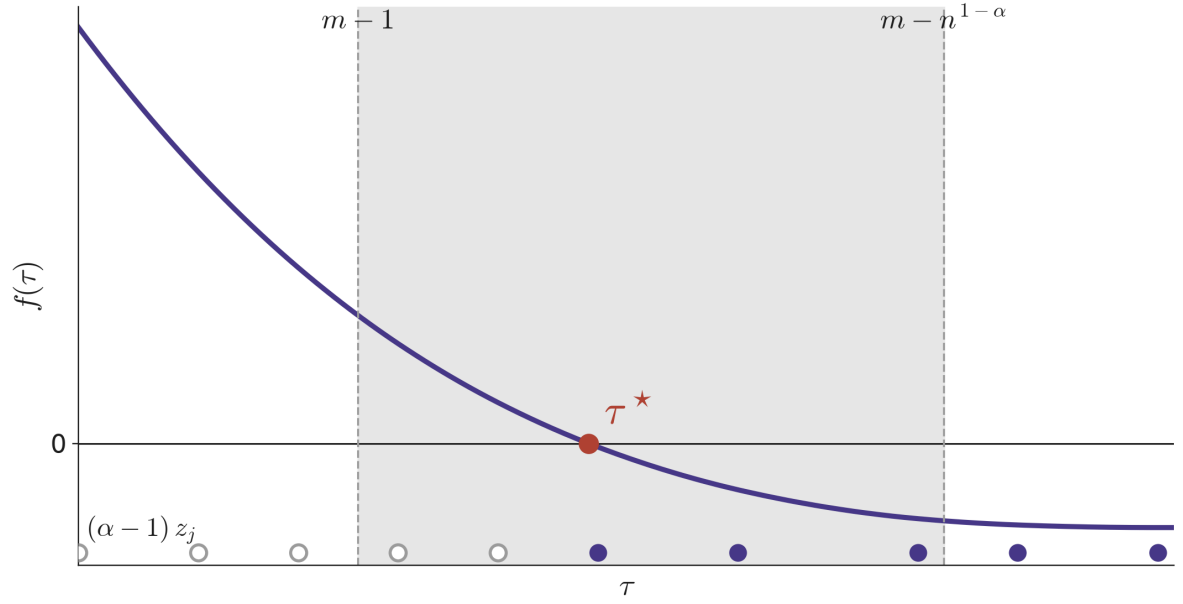
$$f(\tau) := \sum_j p_j - 1$$

$$:= \sum_j \max(0, (\alpha - 1)\boldsymbol{z} - \tau)^{\frac{1}{\alpha-1}} - 1 \quad \longrightarrow \quad \boldsymbol{p}^\star = \max(0, (\alpha - 1)\boldsymbol{z} - \tau^\star)^{\frac{1}{\alpha-1}}$$

with $f(\tau^\star) = 0$

- Solve for $f(\tau) = 0 \rightarrow$ root-finding!
- $f(\tau)$ is *strictly* convex for 1 < α ≤ 2 (unique root)
- One evaluation of $f(\tau)$ = one pass over K
- Bisection:

$$\underbrace{\max_j z_j - \frac{1}{\alpha-1}}_{\text{One-hot solution}} \leq \tau < \underbrace{\max_j z_j}_{\substack{\text{Fully dense case} \\ \text{And every score the same}}}$$



~20 iterations

⚡AdaSplash

- $f(\tau)$ is m times differentiable, whenever $\alpha < \frac{m+1}{m}$

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

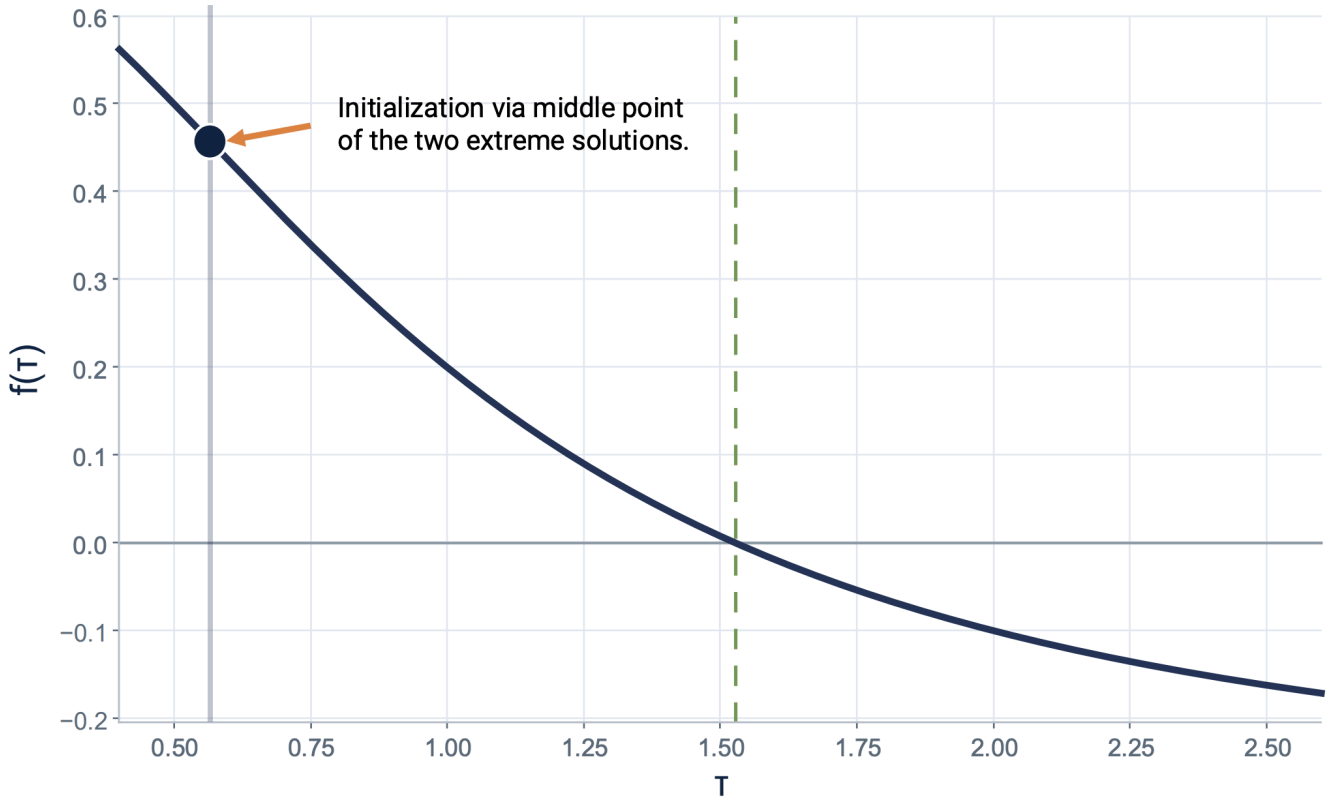
$$f'(\tau) = -\frac{1}{\alpha-1}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{2-\alpha}{\alpha-1}}$$

$$f''(\tau) = \frac{2-\alpha}{(\alpha-1)^2}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{3-2\alpha}{\alpha-1}}$$

- We can use high-order methods such as Newton, Halley

  → They can overshoot the root 😕

- Key idea: keep update within the Bisection bounds!



⚡AdaSplash

- $f(\tau)$ is m times differentiable, whenever $\alpha < \frac{m+1}{m}$

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

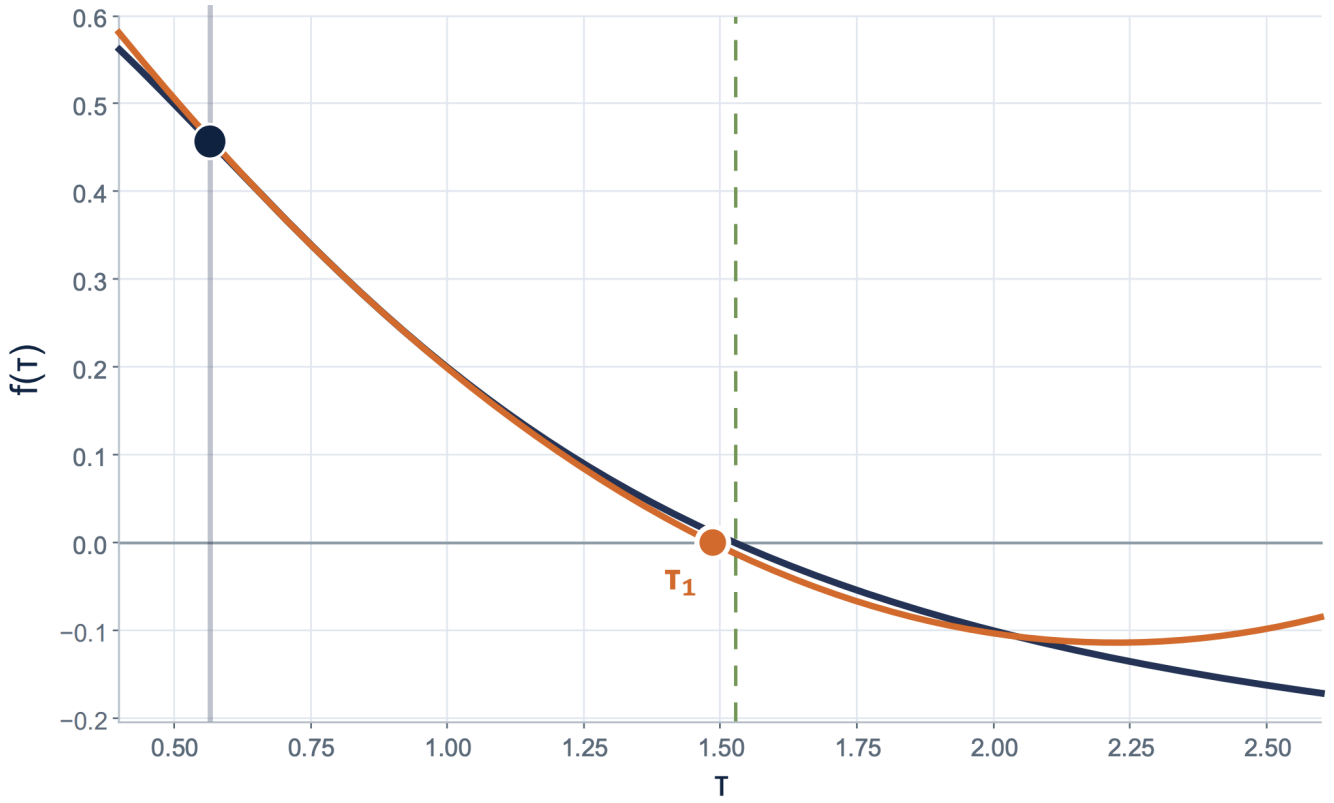
$$f'(\tau) = -\frac{1}{\alpha-1}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{2-\alpha}{\alpha-1}}$$

$$f''(\tau) = \frac{2-\alpha}{(\alpha-1)^2}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{3-2\alpha}{\alpha-1}}$$

- We can use high-order methods such as Newton, Halley

  → They can overshoot the root 😕

- Key idea: keep update within the Bisection bounds!



⚡AdaSplash

- $f(\tau)$ is m times differentiable, whenever $\alpha < \frac{m+1}{m}$

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

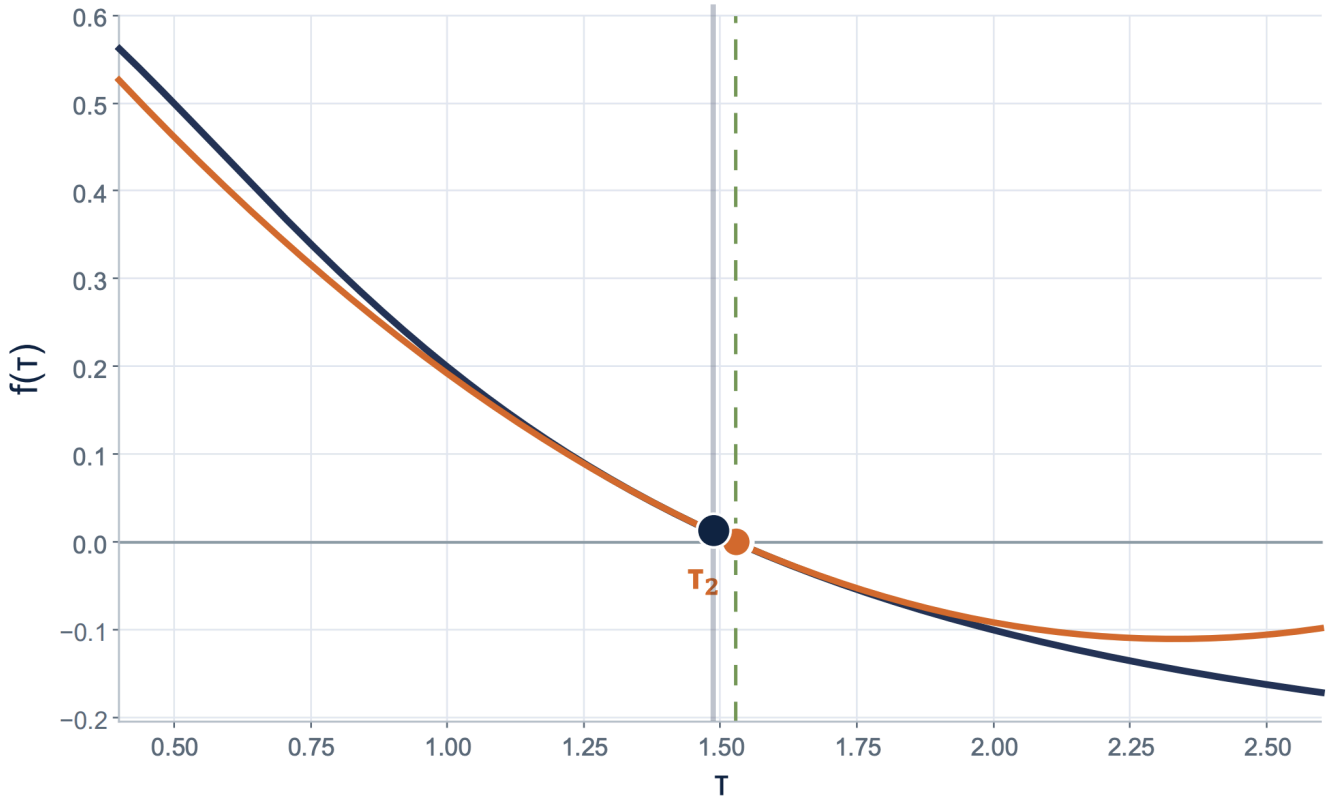
$$f'(\tau) = -\frac{1}{\alpha-1}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{2-\alpha}{\alpha-1}}$$

$$f''(\tau) = \frac{2-\alpha}{(\alpha-1)^2}\sum_i [(\alpha-1)z_i - \tau]_+^{\frac{3-2\alpha}{\alpha-1}}$$

- We can use high-order methods such as Newton, Halley

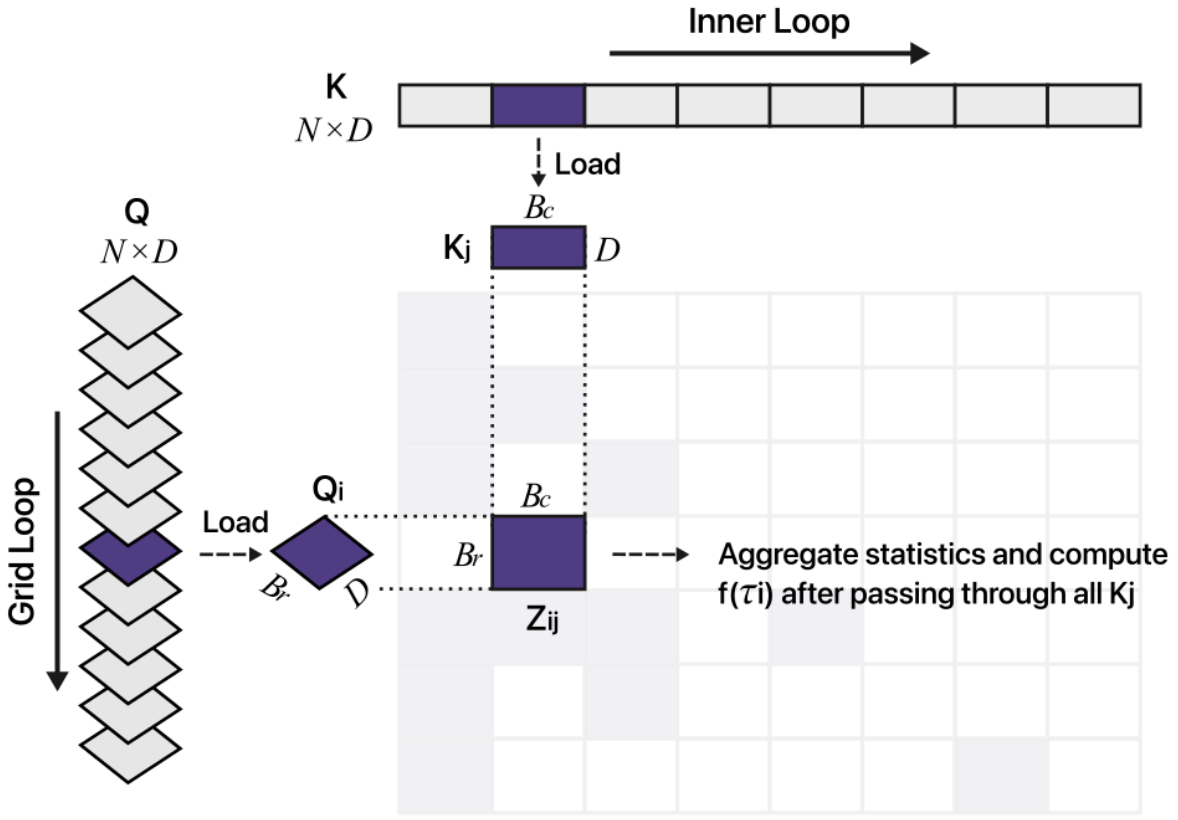
  → They can overshoot the root 😕

- Key idea: keep update within the Bisection bounds!

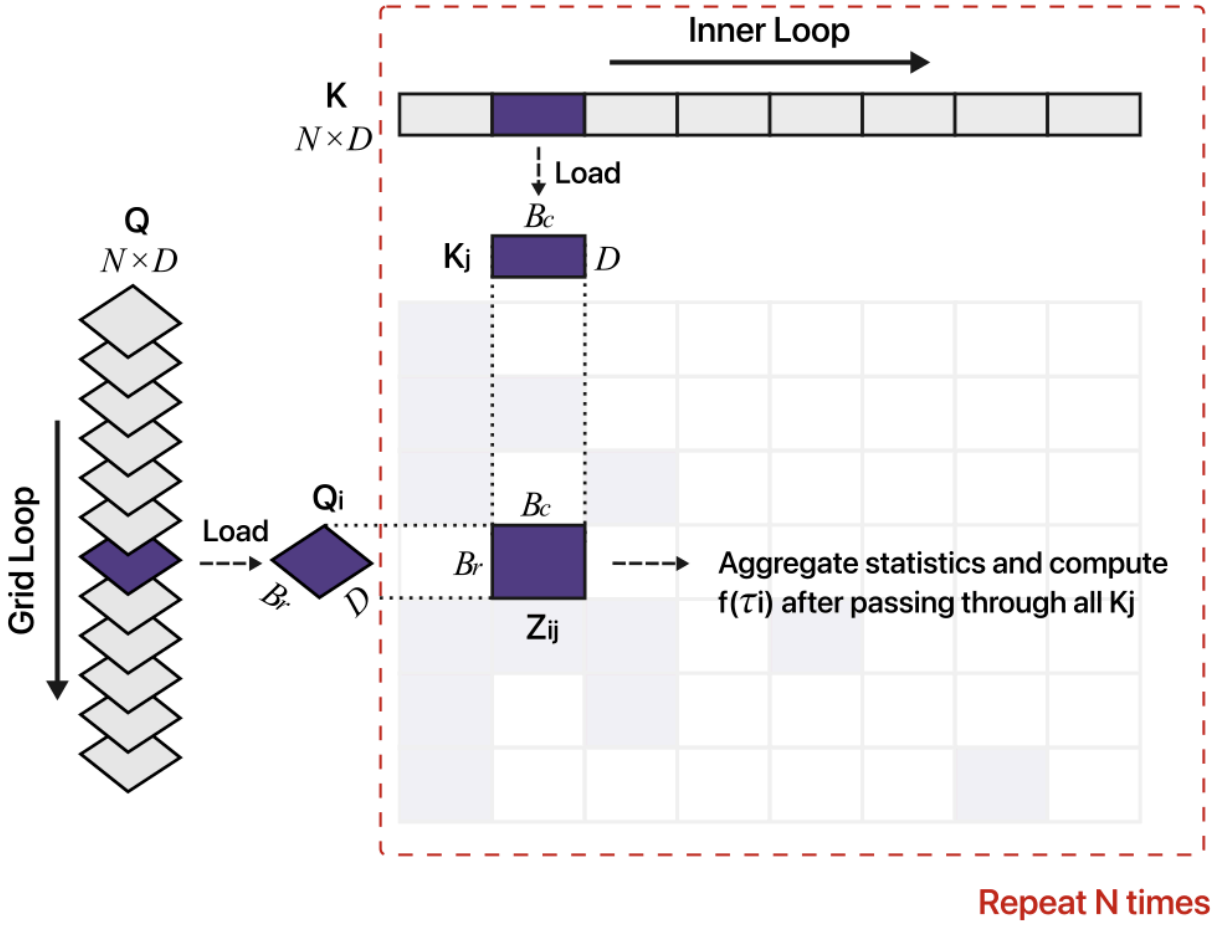


~3-4 iterations!

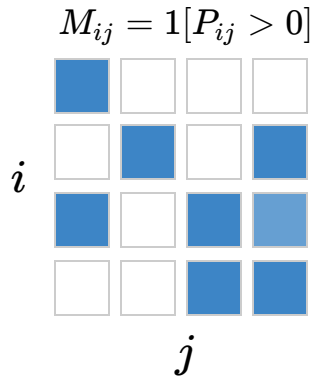
⚡AdaSplash


Inner Loop
K
$N \times D$
Load
B_c
Kj
D
Q
$N \times D$
Grid Loop
Load
Qi
B_r
D
B_c
B_r
Zij
Aggregate statistics and compute
f(τi) after passing through all Kj

⚡AdaSplash


Inner Loop
K
$N \times D$
Load
B_c
Kj
D
Q
$N \times D$
Grid Loop
Load
Qi
B_r
D
B_c
B_r
Zij
Aggregate statistics and compute
f(τi) after passing through all Kj
Repeat N times

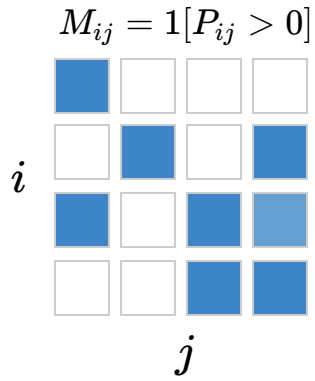
⚡AdaSplash



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

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

⚡AdaSplash



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

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

⚡ Efficient output

$$\boldsymbol{O}_j = \sum_{i \in \mathcal{V}_j} \boldsymbol{P}_i^{(j)} \cdot \boldsymbol{V}_i$$

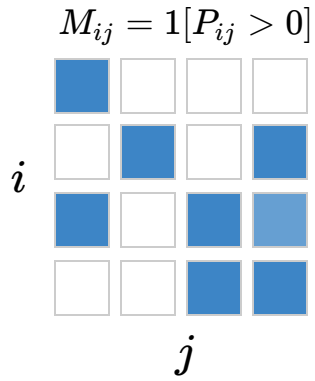
⚡ Efficient backward pass!

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

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

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

⚡AdaSplash



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

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

⚡ Efficient output

$$\boldsymbol{O}_j = \sum_{i \in \mathcal{V}_j} \boldsymbol{P}_i^{(j)} \cdot \boldsymbol{V}_i$$

⚡ Efficient backward pass!

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

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

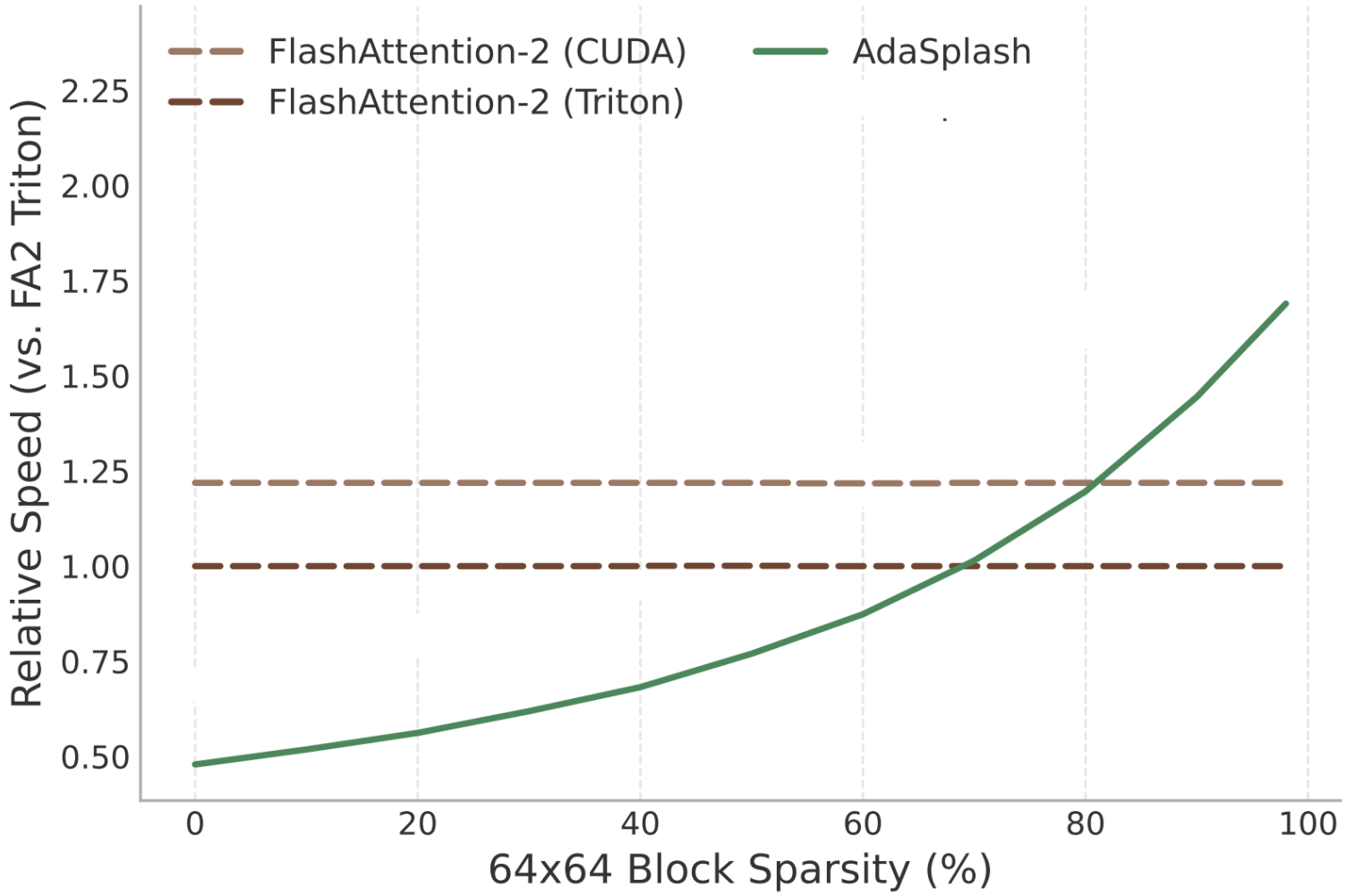
$$\boldsymbol{dV}_j = \sum_{i \in \mathcal{K}_j} \boldsymbol{P}_i^{(j)} \cdot \boldsymbol{dO}_i$$

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

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 (CUDA)
FlashAttention-2 (Triton)
AdaSplash
Relative Speed (vs. FA2 Triton)
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
20
40
60
80
100
64x64 Block Sparsity (%)

⚡⚡ 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

📌 ICML 2026

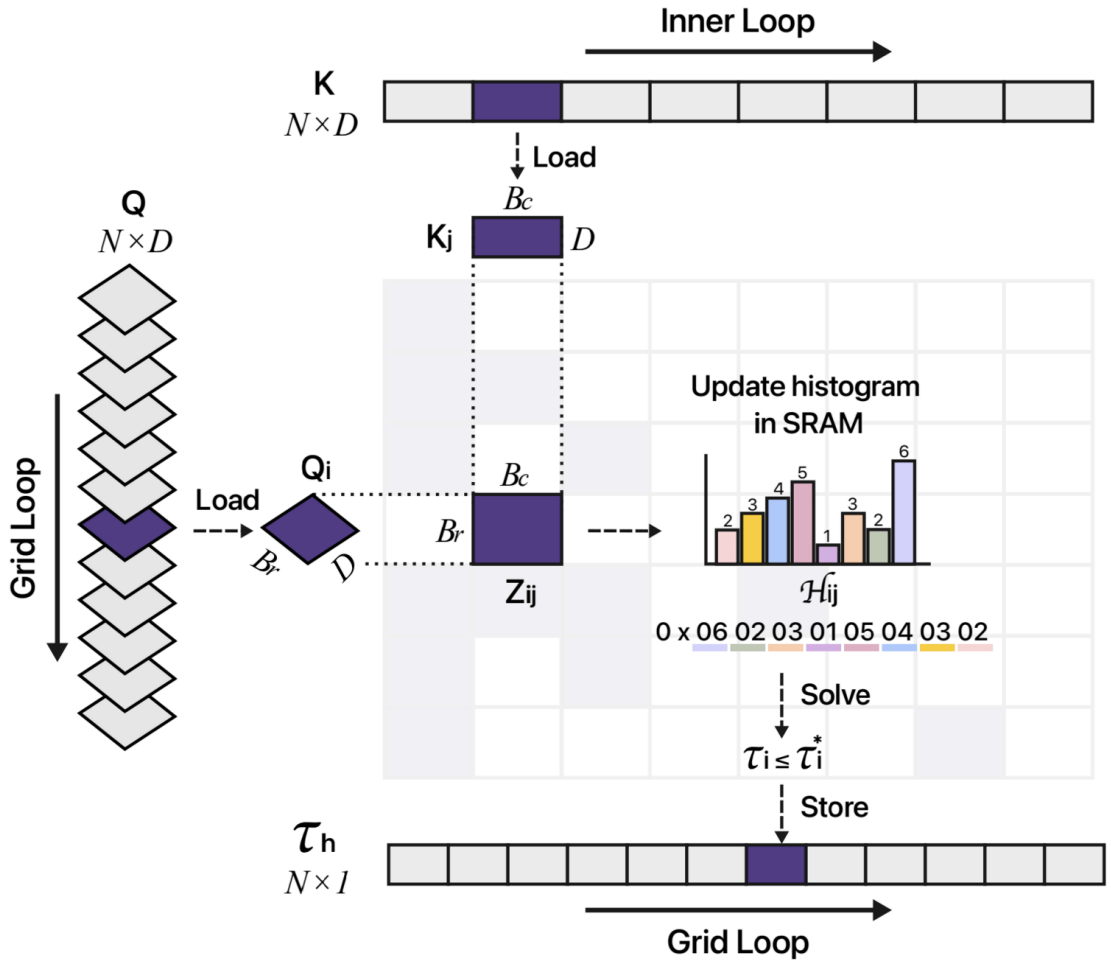
📄 arxiv.org/abs/2604.15180

💻 github.com/deep-spin/adasplash

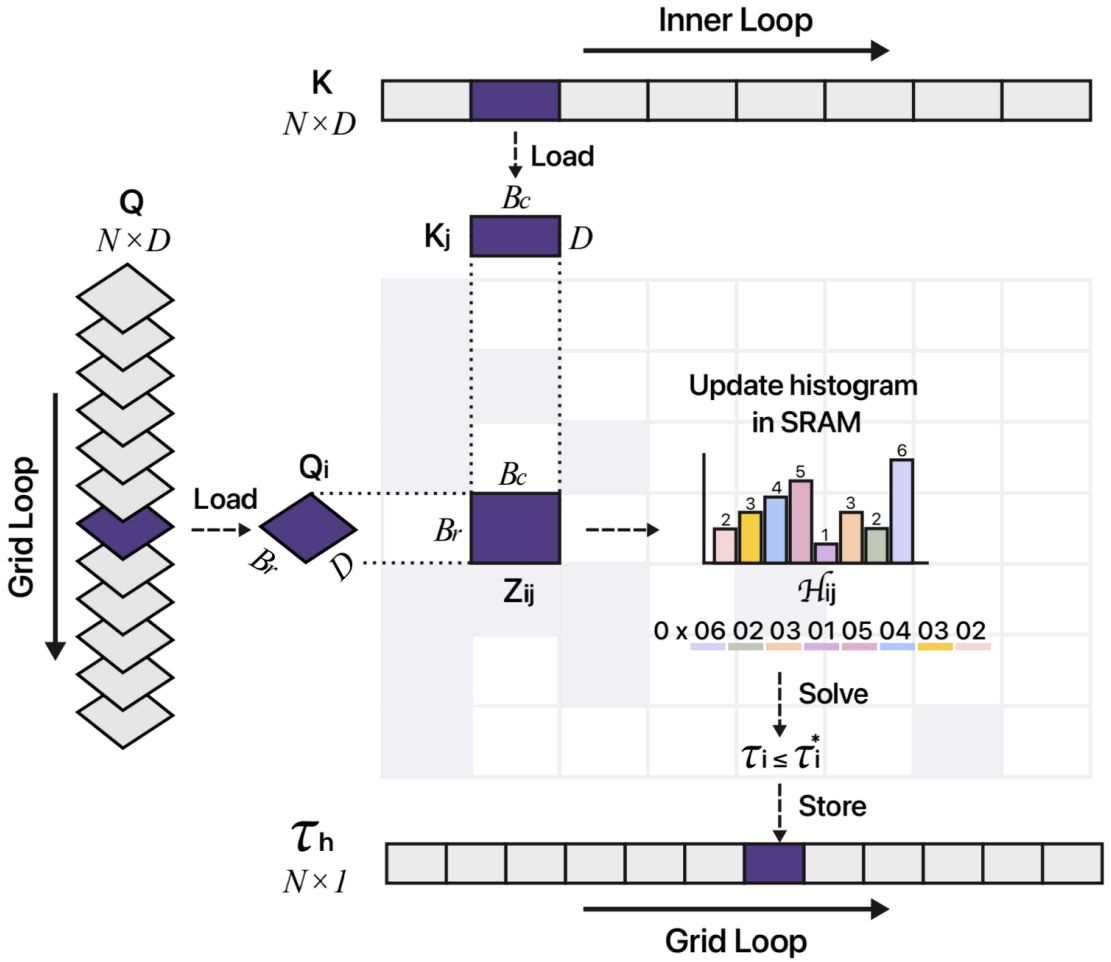
```python
# pip install adasplash
from adasplash import adasplash_v2

y = adasplash_v2(q, k, v, alpha=1.5, niter=3, is_causal=True)
```

⚡⚡ AdaSplash-2: Faster Differentiable Sparse Attention



⚡⚡ AdaSplash-2: Faster Differentiable Sparse Attention



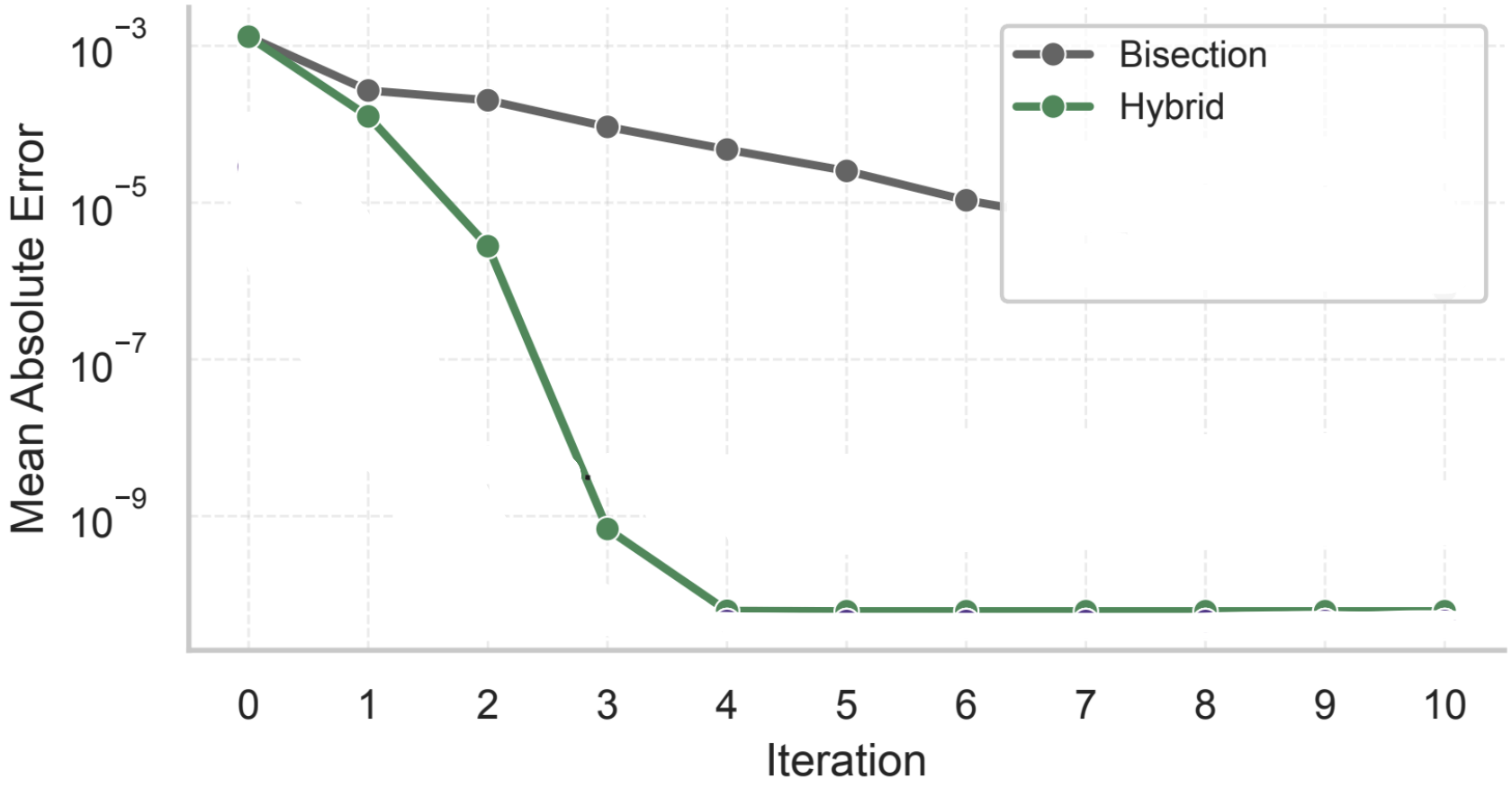
$$\bar{z}_j = (\alpha - 1)z_j - (m - 1) \quad \Rightarrow \quad \max_j \bar{z}_j = 1, \quad \tau^\star \in [0, 1)$$

$$H_k = \left|\{\, j : \min(\lfloor B\bar{z}_j \rfloor,\, B-1) = k \,\}\right|, \quad k = 0, \ldots, B-1$$

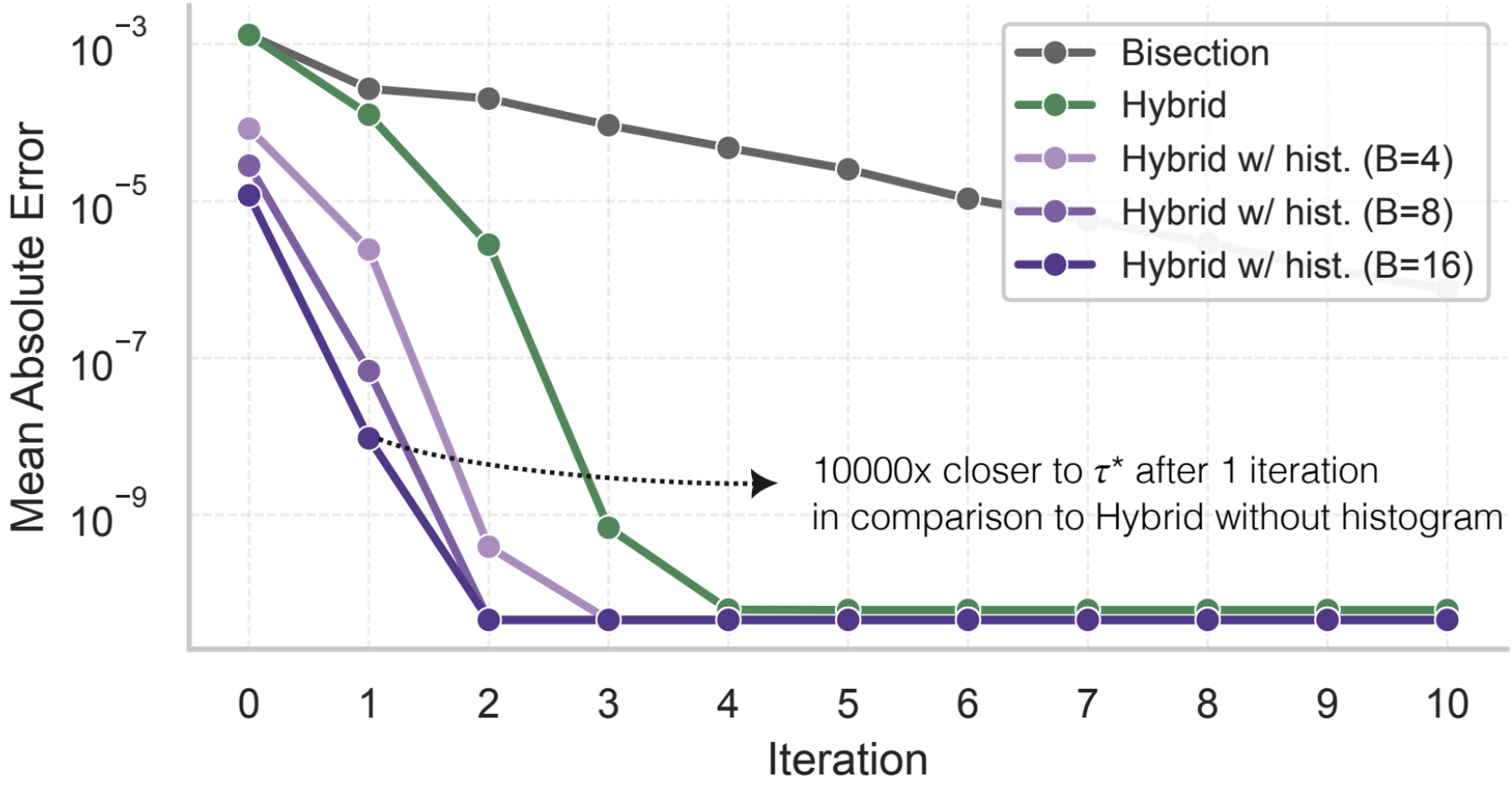
$$f_h(\tau) := \sum_{k=0}^{B-1} H_k \left[\frac{k}{B} - \tau\right]_+^{\frac{1}{\alpha-1}} - 1$$

- $B = 8$ bins, packed in one 64-bit integer, *in registers*!
  - Single cycle, super fast ⚡
- Solve in $O(B)$, not $O(n)$
  - the bins are already sorted -> closed form for τ_h

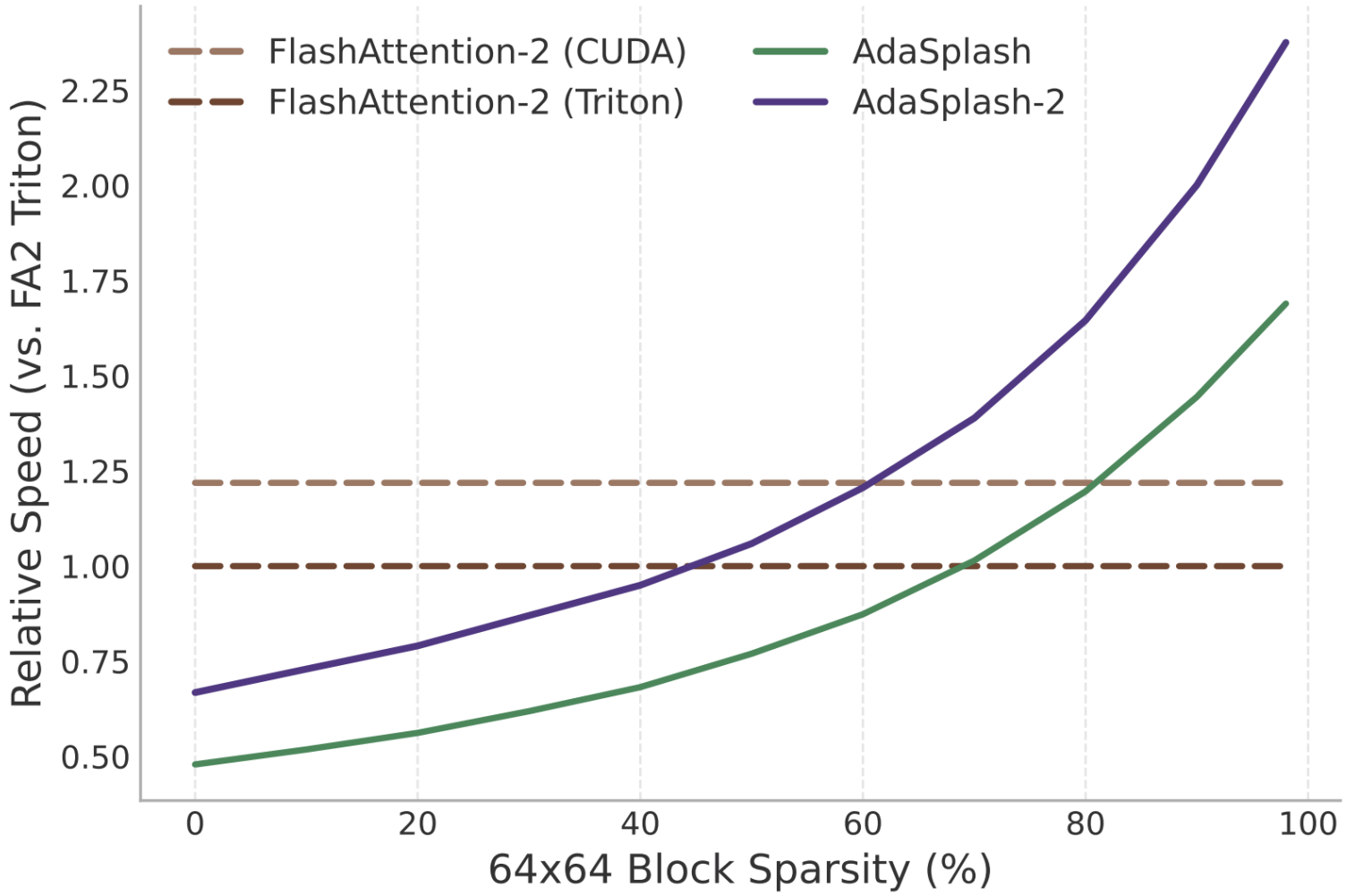
⚡⚡ AdaSplash-2: Faster Differentiable Sparse Attention



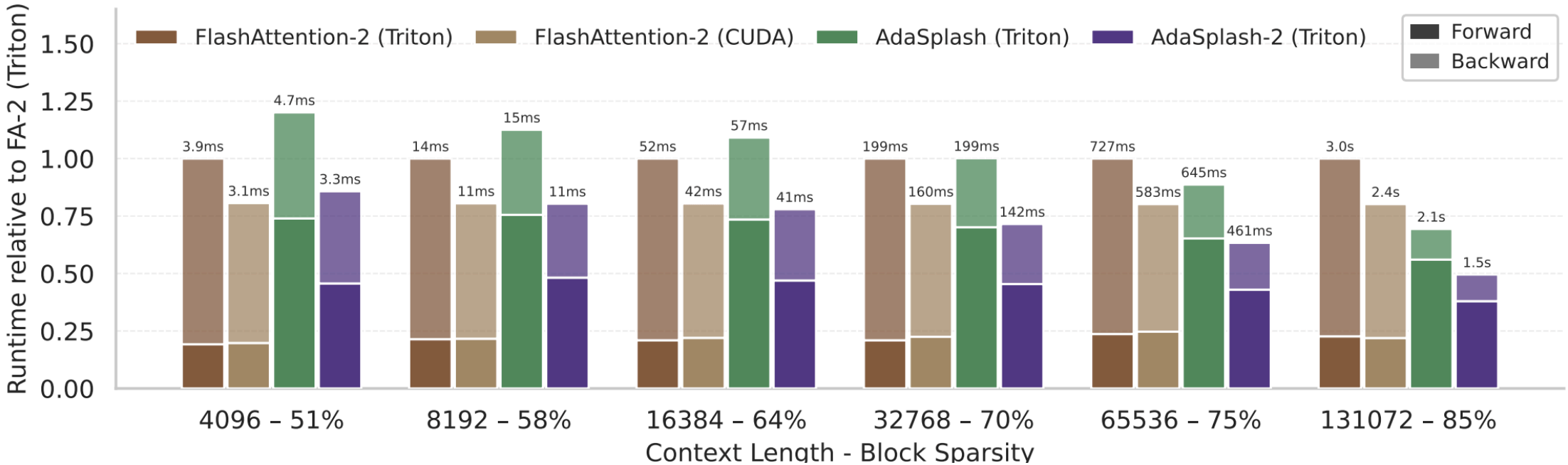
⚡⚡ AdaSplash-2: Faster Differentiable Sparse Attention



⚡AdaSplash-2:


FlashAttention-2 (CUDA)
FlashAttention-2 (Triton)
AdaSplash
AdaSplash-2
Relative Speed (vs. FA2 Triton)
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
20
40
60
80
100
64x64 Block Sparsity (%)

⚡AdaSplash-2:


FlashAttention-2 (Triton)
FlashAttention-2 (CUDA)
AdaSplash (Triton)
AdaSplash-2 (Triton)
Forward
Backward
Runtime relative to FA-2 (Triton)
1.50
1.25
1.00
0.75
0.50
0.25
0.00
3.9ms
3.1ms
4.7ms
3.3ms
14ms
11ms
15ms
11ms
52ms
42ms
57ms
41ms
199ms
160ms
199ms
142ms
727ms
583ms
645ms
461ms
3.0s
2.4s
2.1s
1.5s
4096 – 51%
8192 – 58%
16384 – 64%
32768 – 70%
65536 – 75%
131072 – 85%
Context Length - Block Sparsity

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?

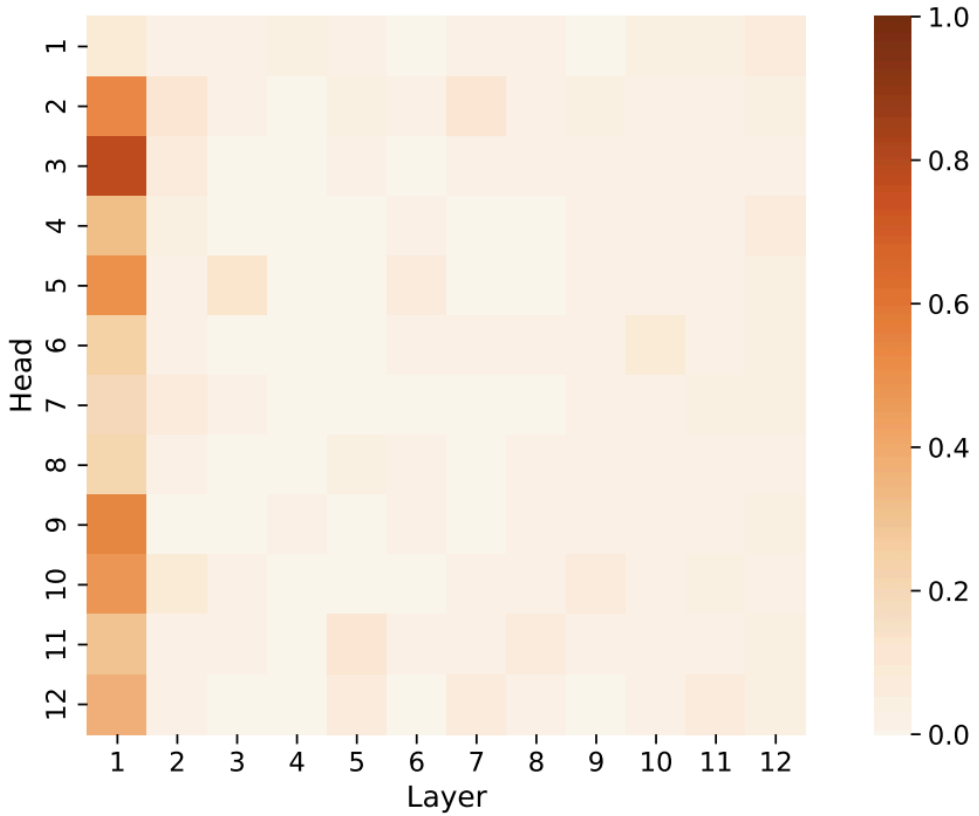


Figure 4. Ratio of non-zero attention scores for GPT-2 ($\alpha = 1.5$).

AdaSplash-3 (preview)

- In collaboration with NVIDIA
- From Triton to CuteDSL (Hopper) and Cutile (Blackwell) + novel algorithmic ideas
- Will be previewed at GTC in Paris in two days

N	ms	vs FA3	vs TRITON
2,048	1.44	0.78×	1.58×
4,096	4.08	0.94×	2.00×
8,192	11.92	1.19×	2.55×
16,384	36.65	1.47×	3.21×

Tested on random inputs.
Trained models are far sparser, especially on longer contexts

Long-context Generalization with Sparse Attention

w/



Pavlo Vasylenko



Hugo Pitorro



André F. T. Martins

📌 ICLR 2026

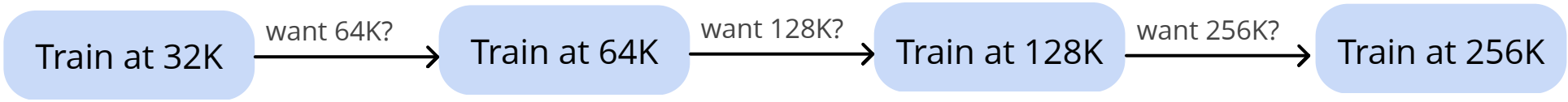
📄 arxiv.org/abs/2506.16640

💻 github.com/deep-spin/asentmax

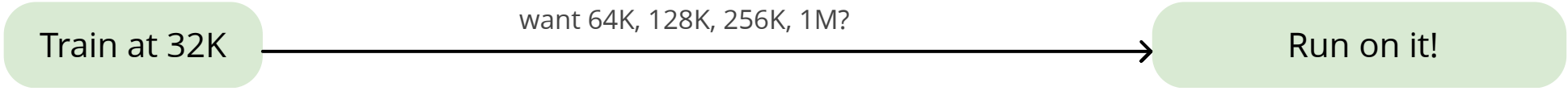
What happens with attention on sequences longer than those seen during training?

Long-context Generalization

path of pain 😩



green path 🌿



arxiv.org/abs/2506.16640

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:*

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

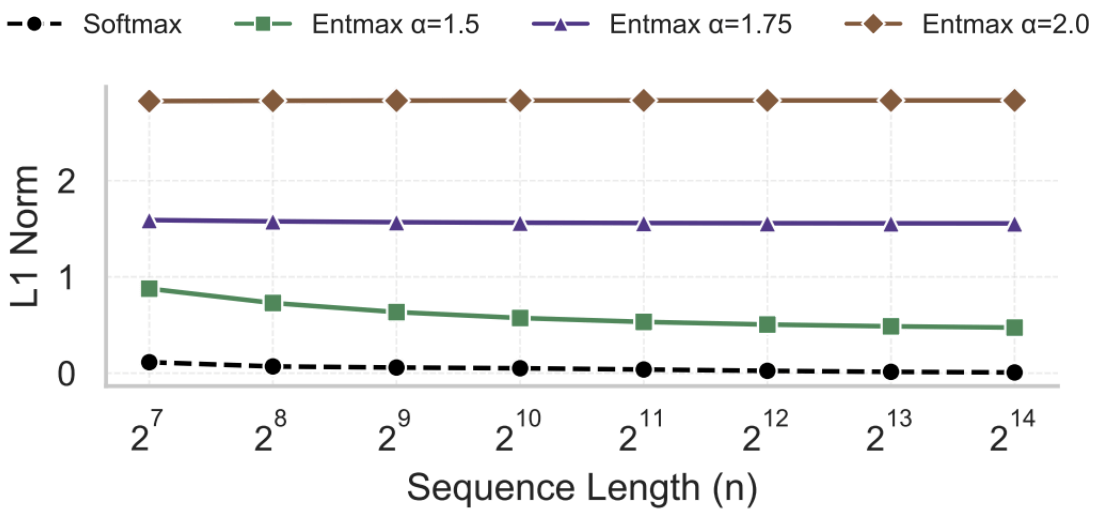
Proposition 2 (Representational Preservation and Reduced Gradient Paths). *Let $\alpha > 1$. Consider a depth-L transformer with residual connections and attention weights given by α-entmax. Suppose each attention distribution has support size at most s with $s \ll n$. Then:*

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 $n \to \infty$. In contrast, Barbero et al. (2024) shows that for softmax attention, $\|\boldsymbol{v}_n^{(L)} - \boldsymbol{v}_{n+1}^{*(L)}\|_1 \to 0$ as $n \to \infty$.*

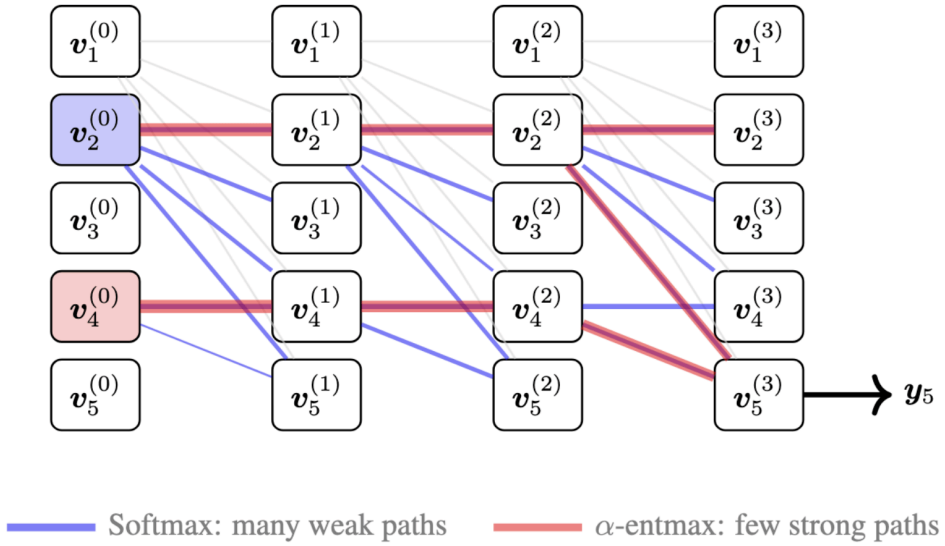
2. *(**Alleviated over-squashing**) The number of effective gradient paths scales as $\mathcal{O}(s^L)$ (rather than $\mathcal{O}(n^L)$ under softmax), alleviating over-squashing by concentrating gradient flow on fewer active paths.*

Theoretical Findings

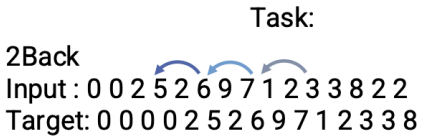
✅ Preserve representations



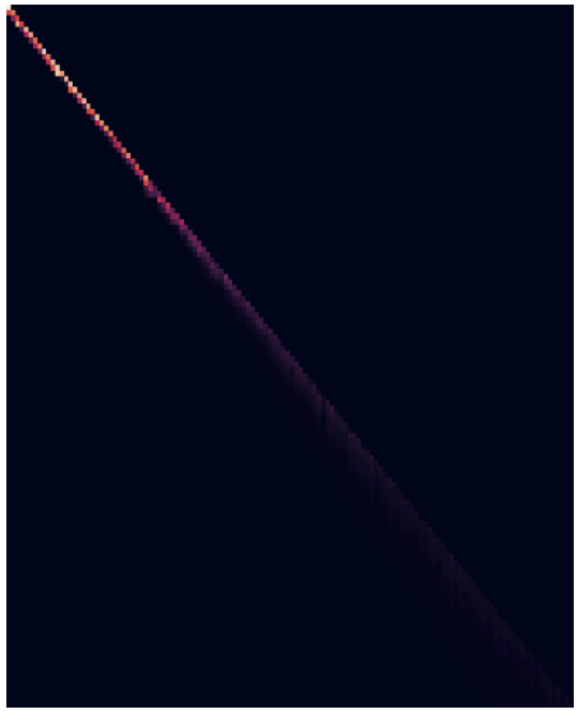
✅ Alleviate over-squashing



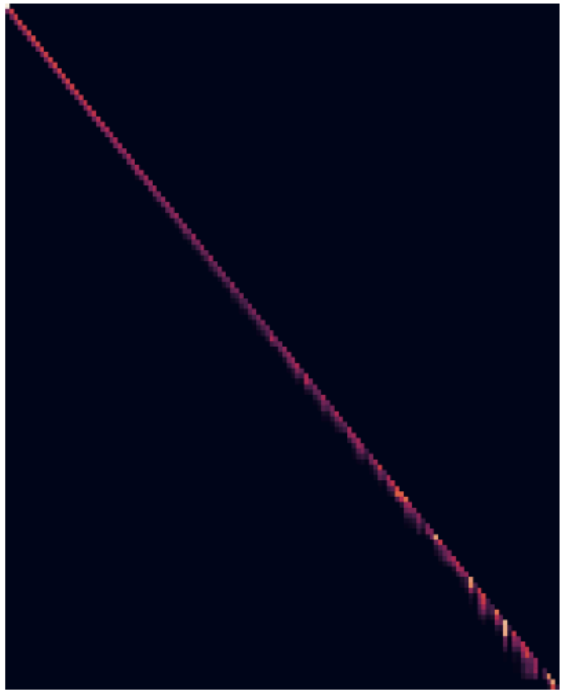
Theoretical Findings



✅ Maintain focus (non-dispersion)



Softmax



Entmax

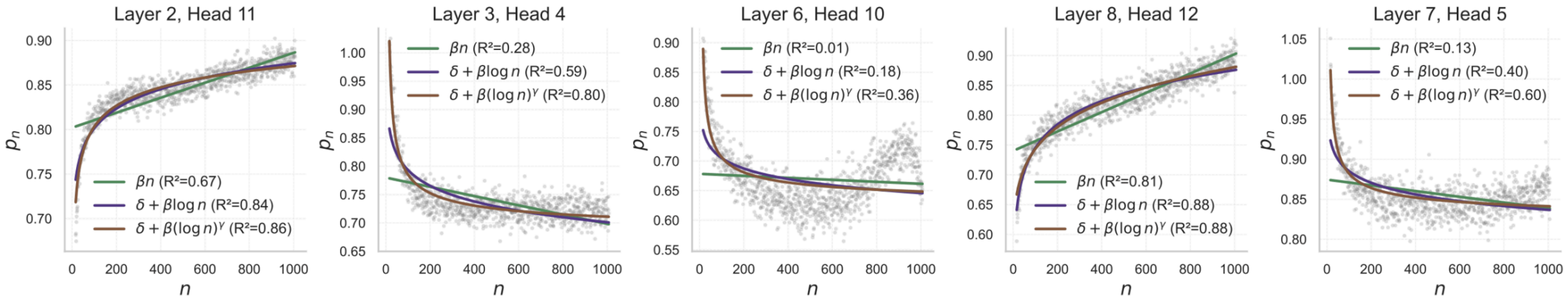
Adaptive-Scalable α-entmax (ASEntmax)

What if entmax ignores *too many* relevant tokens?

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

scaler to automatically adjust sparsity according to the context-length

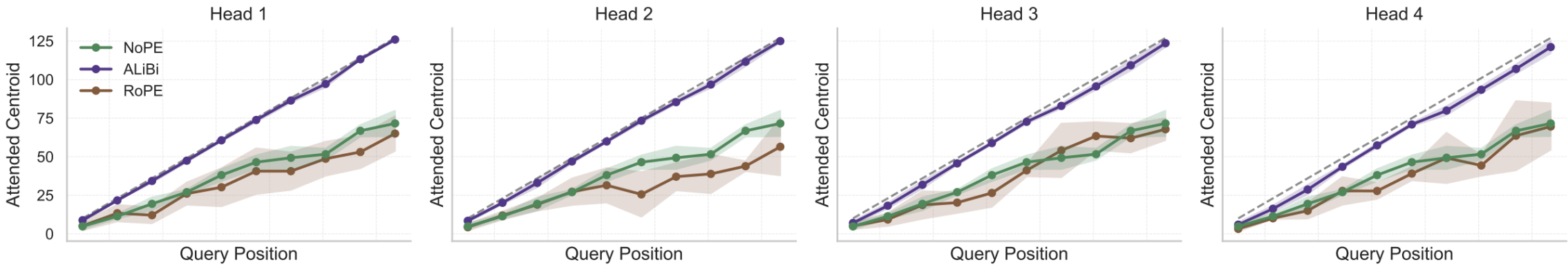
δ, β, γ are *learnable* parameters



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

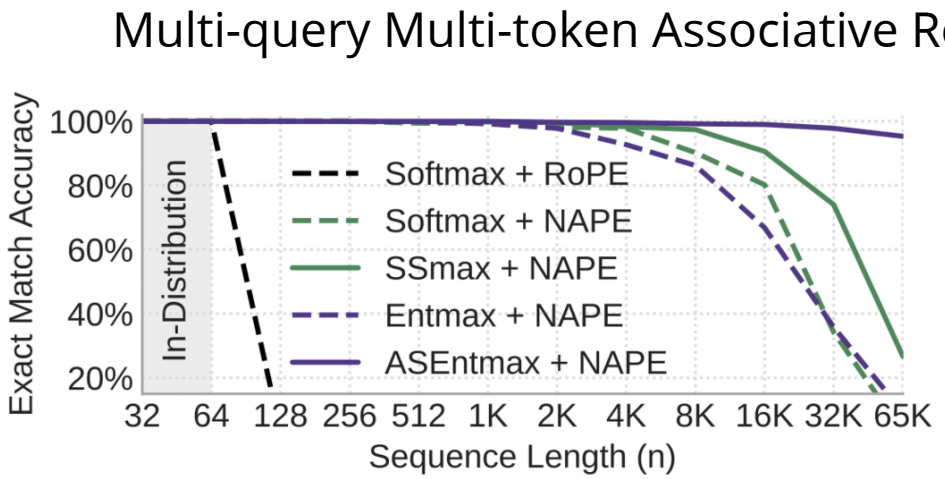


NAPE (NoPE + ALiBi) => half of the heads use NoPE, the other half use ALiBi

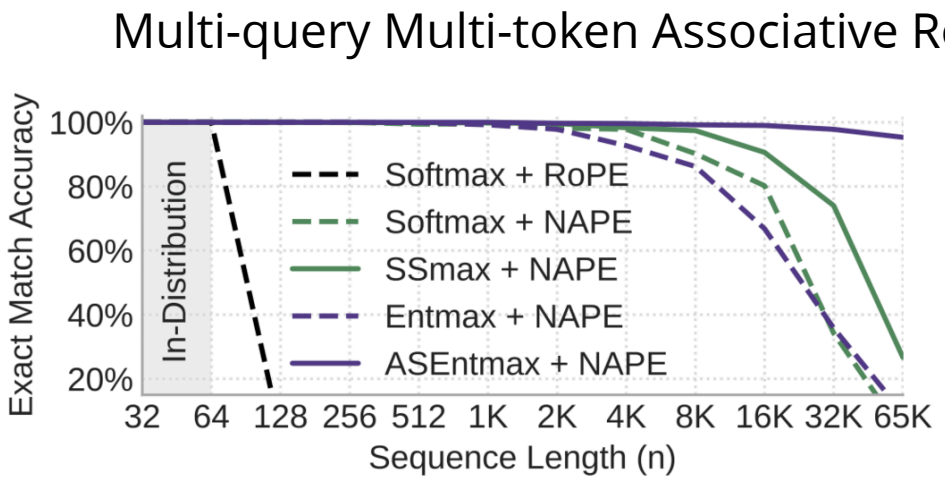
- short-span focus

- semantic-guided attention

Results


Multi-query Multi-token Associative Recall
Exact Match Accuracy
100%
80%
60%
40%
20%
In-Distribution
Softmax + RoPE
Softmax + NAPE
SSmax + NAPE
Entmax + NAPE
ASEntmax + NAPE
32
64
128
256
512
1K
2K
4K
8K
16K
32K
65K
Sequence Length (n)

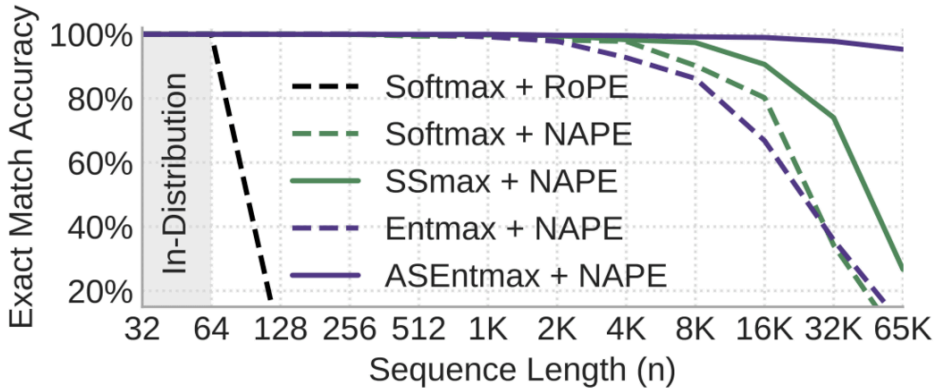
Results



1000x extrapolation!

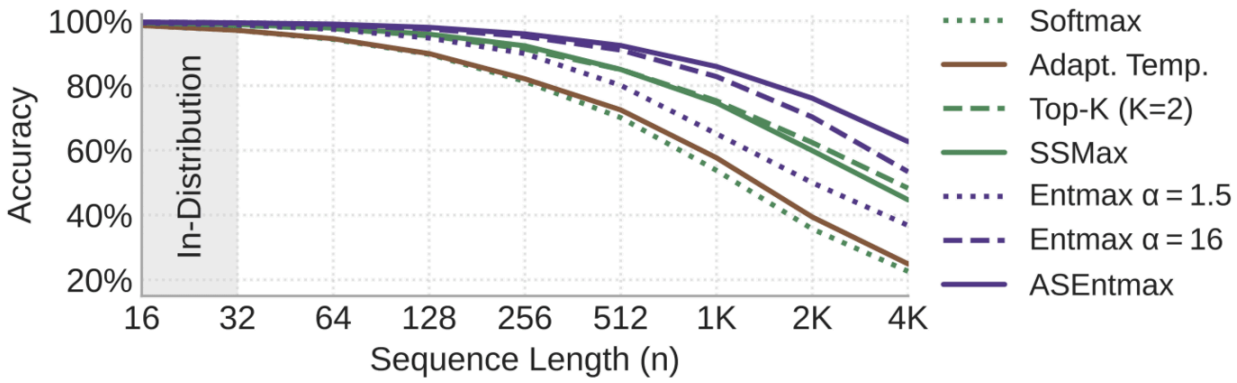
Results

Multi-query Multi-token Associative Recall



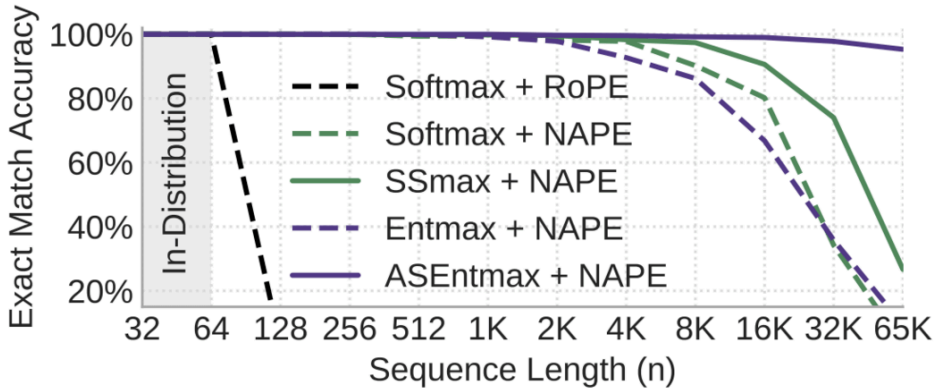
1000x extrapolation!

Max Retrieval



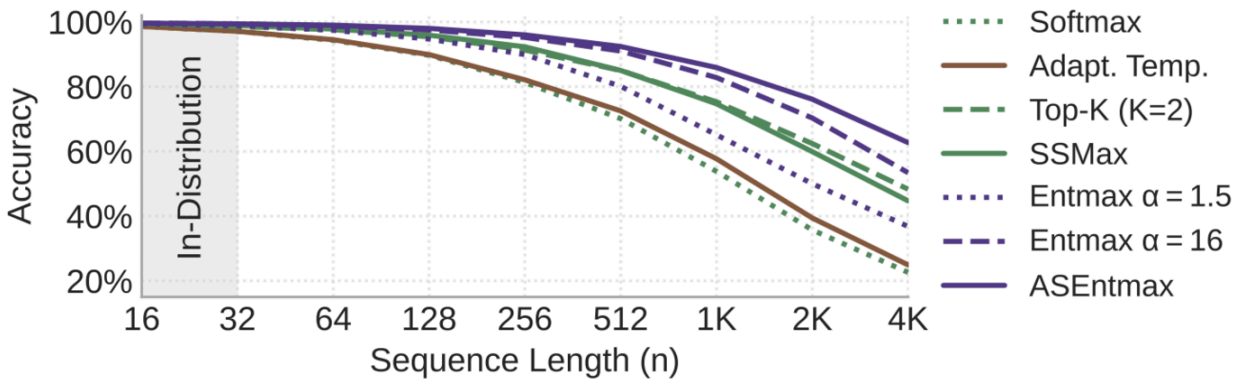
Results

Multi-query Multi-token Associative Recall



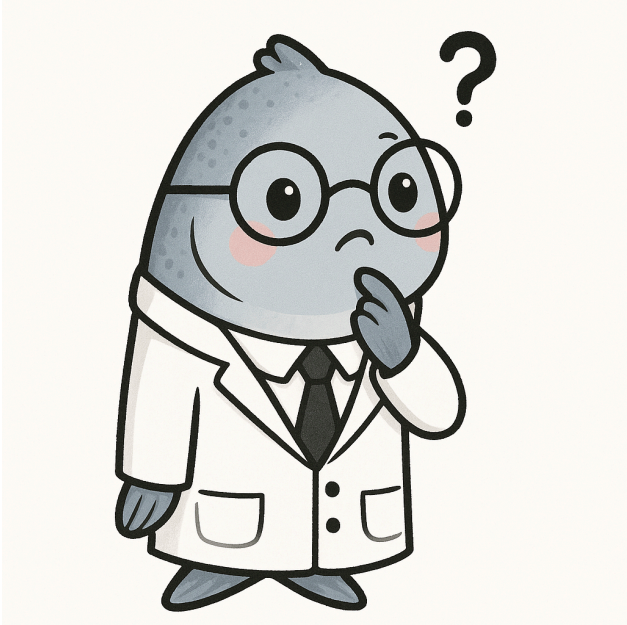
1000x extrapolation!

Max Retrieval

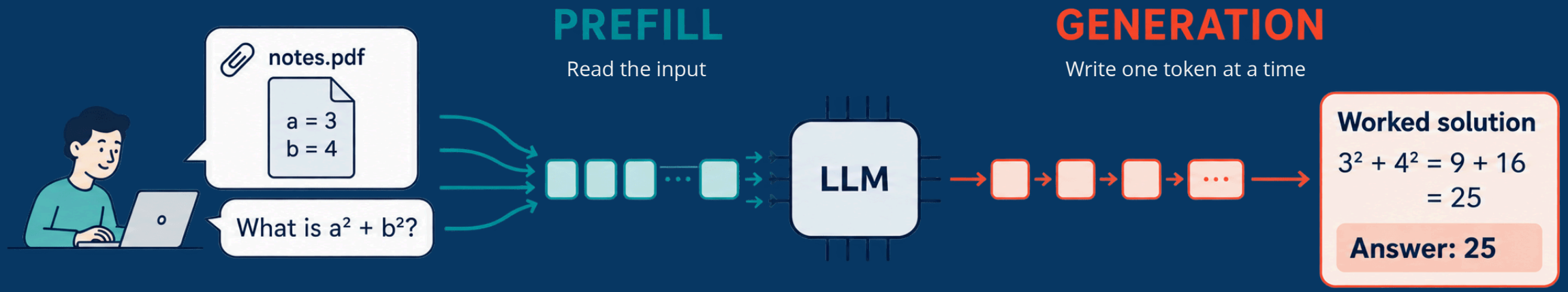


128x extrapolation!

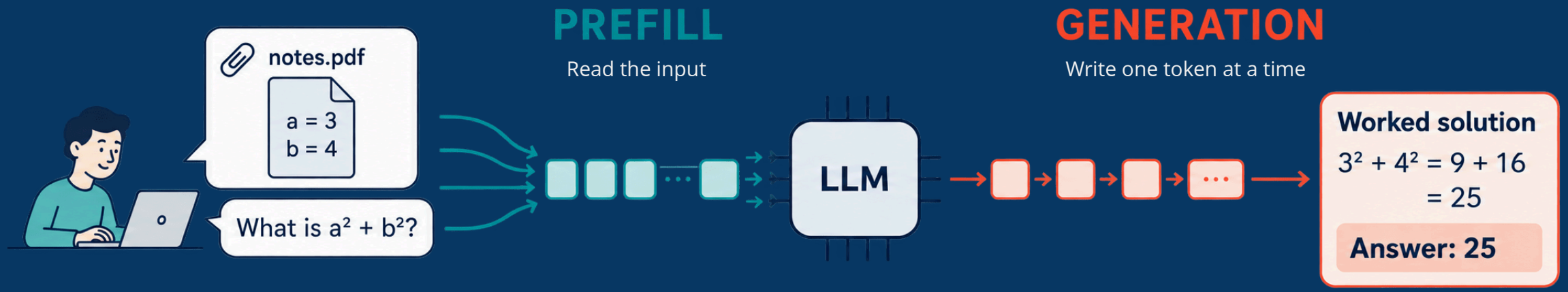
Can we actually accelerate inference?



Inference is a totally different beast



Inference is a totally different beast



compute-bound

most of the time we are doing actual computation

memory-bound

most of the time we are waiting for memory transfers

EntmaxKV

w/



Gonçalo Duarte

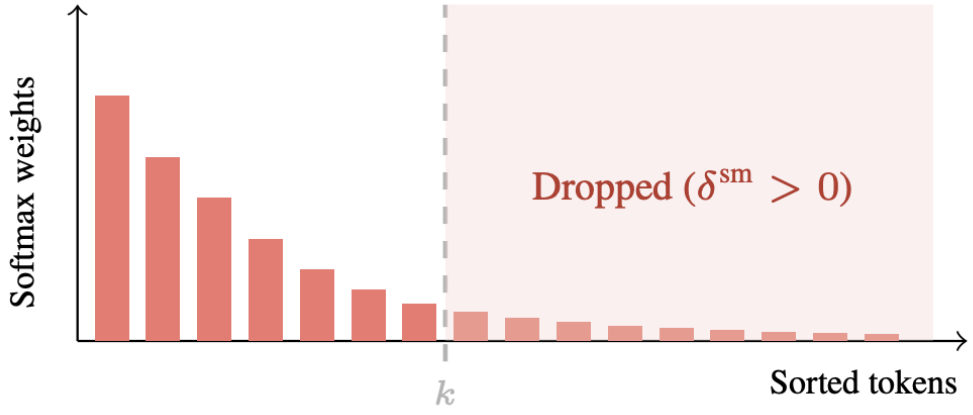


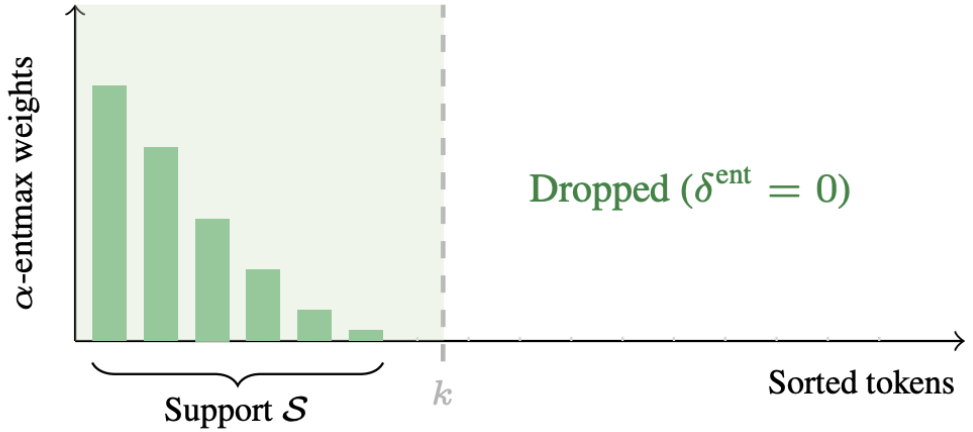
Miguel Couceiro

📌 NeurIPS 2026 Workshop on Long-Context Foundation Models

📄 arxiv.org/abs/2605.21649

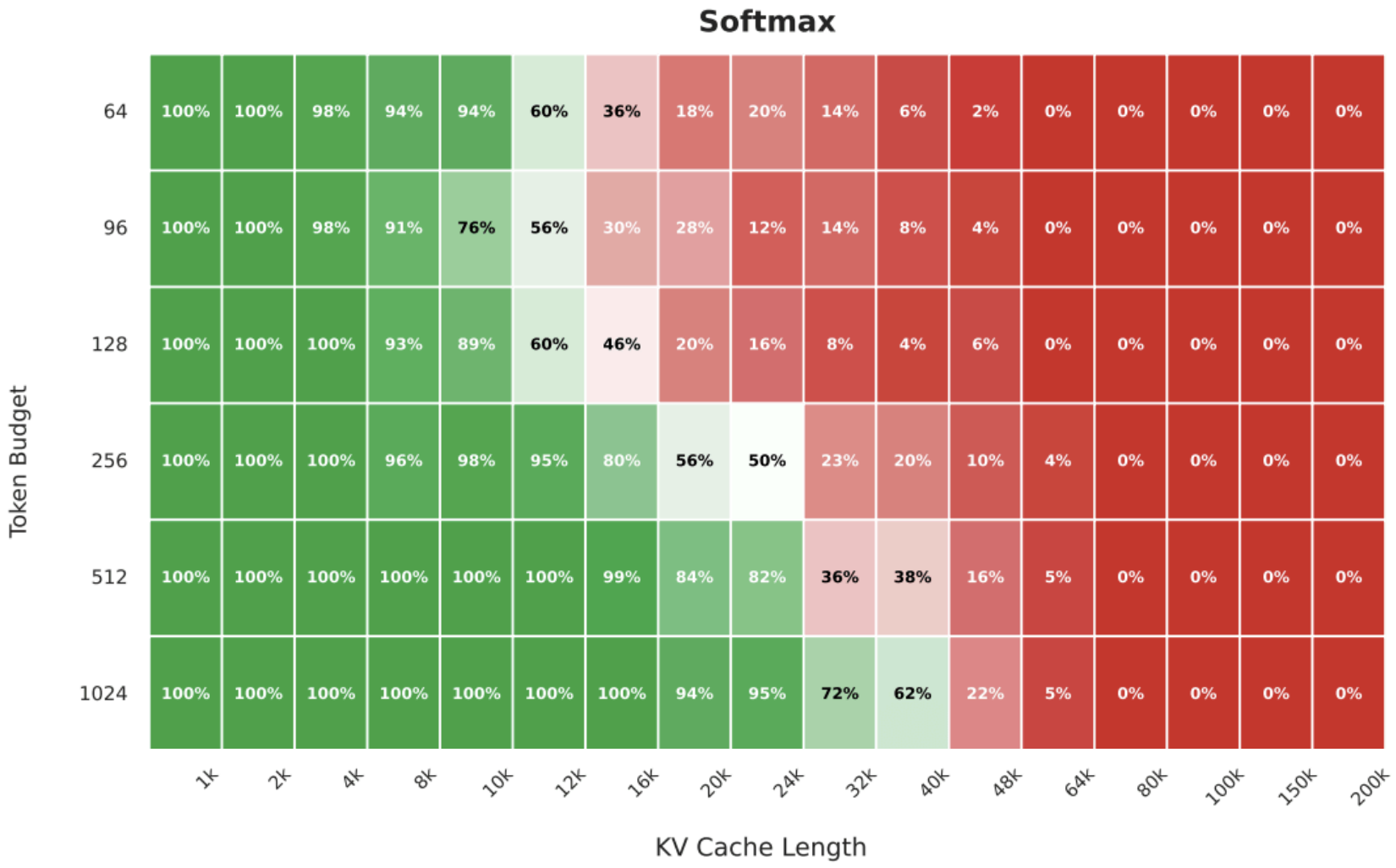
💻 github.com/deep-spin/entmaxkv



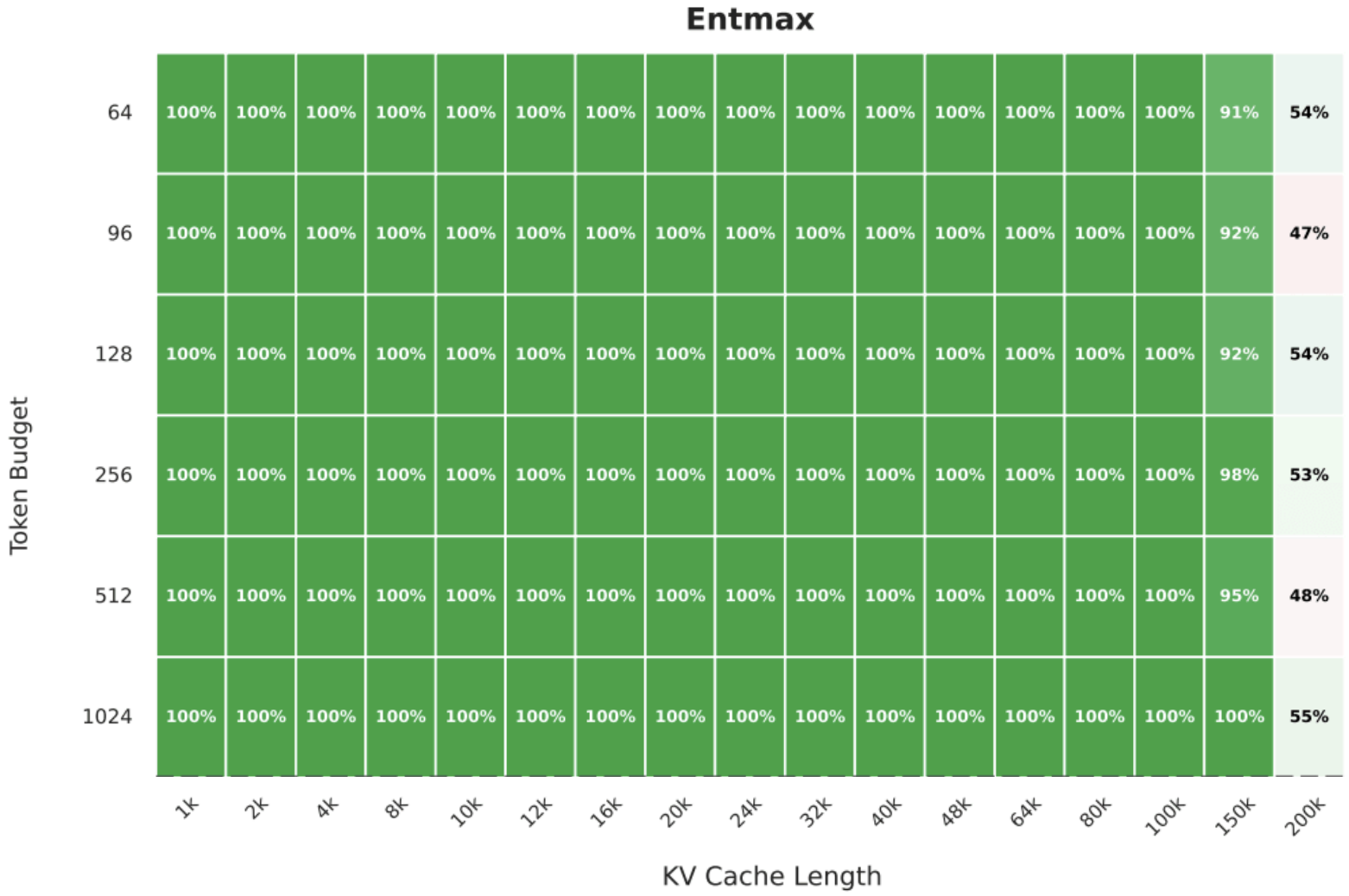


$$\|\boldsymbol{v}_i\|_2 \leq B \;\Rightarrow\; \|\boldsymbol{o} - \tilde{\boldsymbol{o}}\|_2 \leq 2B\,\delta \longrightarrow$$ Lower δ implies a **lower approximation error**

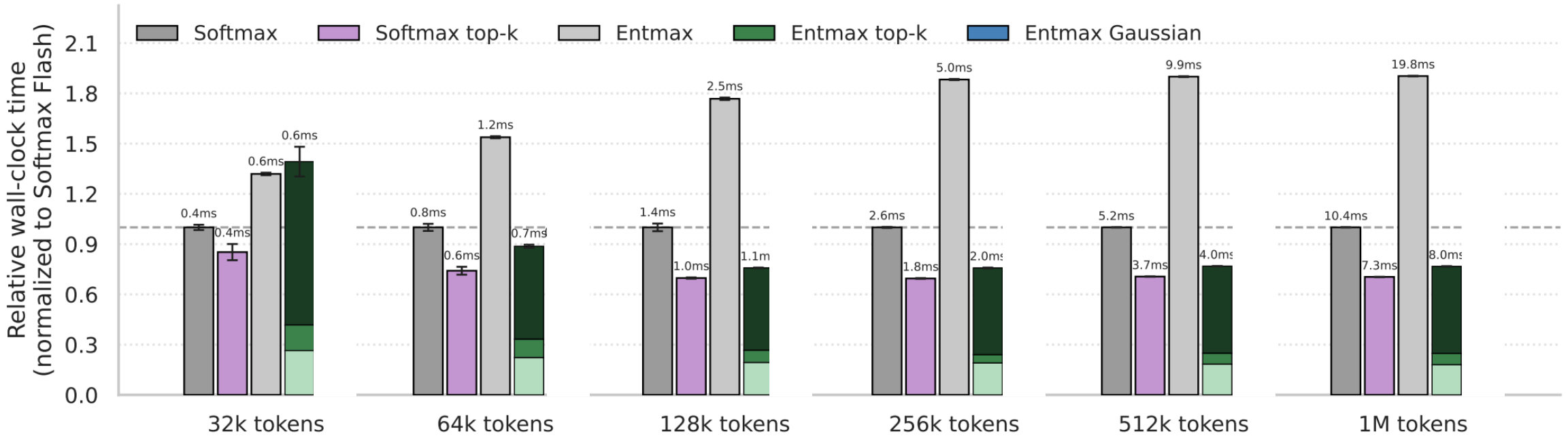
Needle in a Haystack



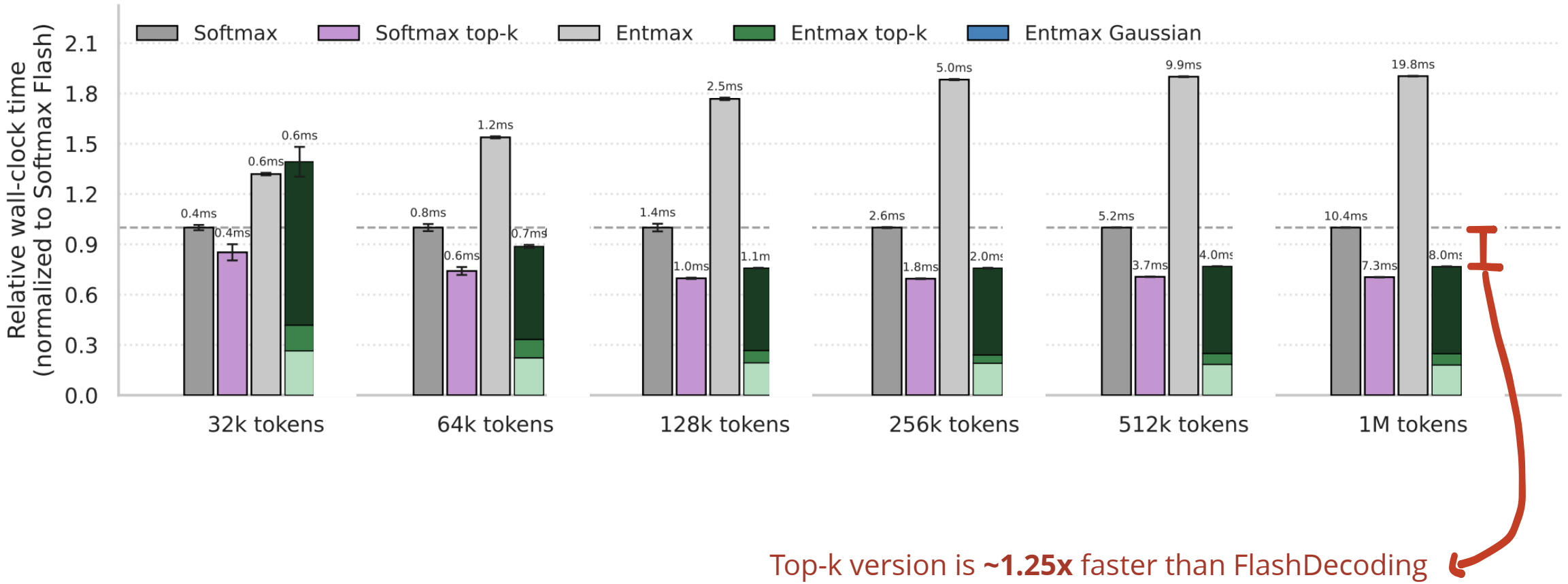
Needle in a Haystack



EntmaxKV

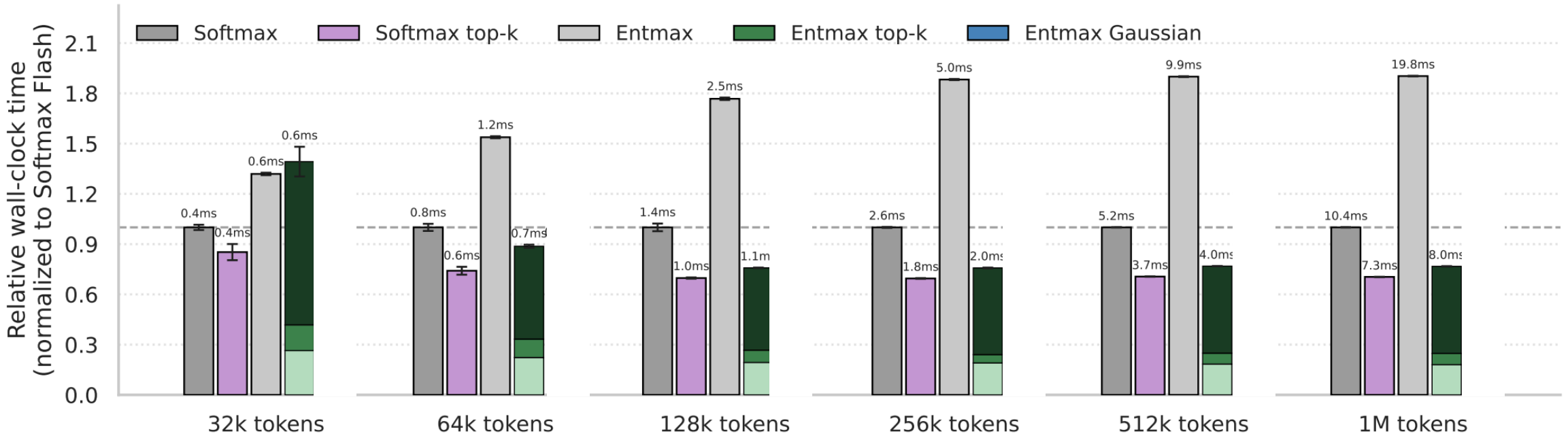

Softmax
Softmax top-k
Entmax
Entmax top-k
Entmax Gaussian
Relative wall-clock time
(normalized to Softmax Flash)
2.1
1.8
1.5
1.2
0.9
0.6
0.3
0.0
0.4ms
0.4ms
0.6ms
0.6ms
0.8ms
0.6ms
1.2ms
0.7ms
1.4ms
1.0ms
2.5ms
1.1m
2.6ms
1.8ms
5.0ms
2.0ms
5.2ms
3.7ms
9.9ms
4.0ms
10.4ms
7.3ms
19.8ms
8.0ms
32k tokens
64k tokens
128k tokens
256k tokens
512k tokens
1M tokens

Efficiency



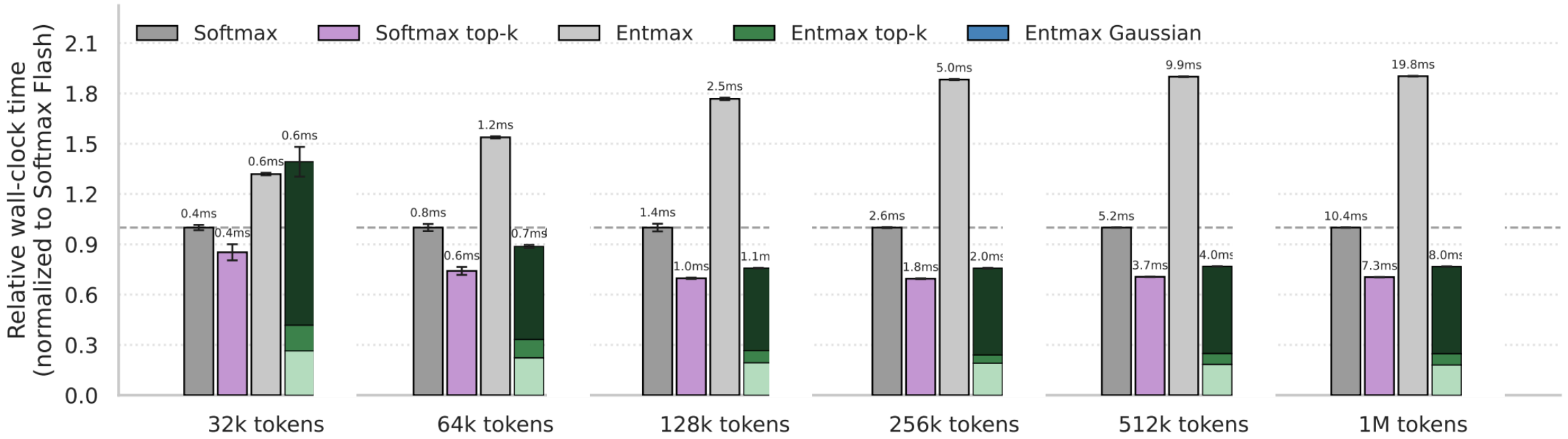
Top-k version is **~1.25x** faster than FlashDecoding

Efficiency



Can we do better?

Efficiency

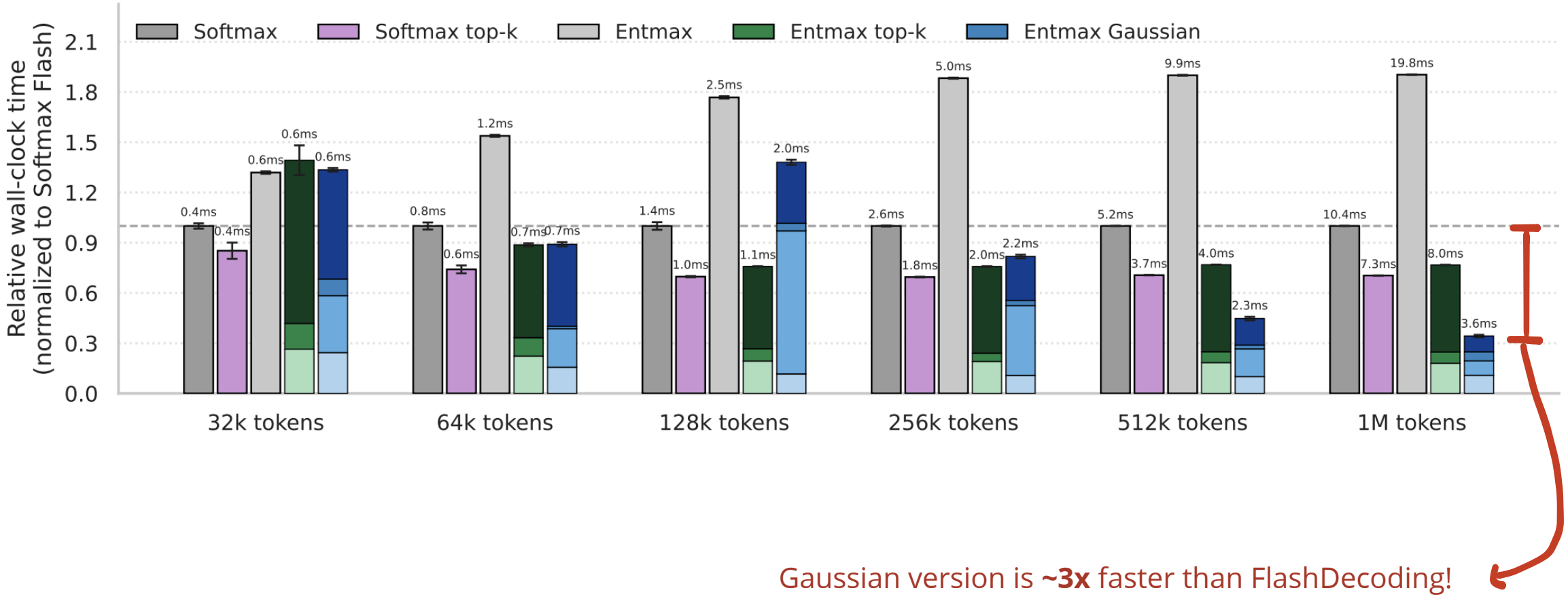


$$S^{(p)} \sim \mathcal{N}(\mu_p, \sigma_p^2): \quad \sum_p |\mathcal{P}_p| \, \mathbb{E}\left[\left[(\alpha-1)S^{(p)} - \hat{\tau}\right]_+^{\frac{1}{\alpha-1}}\right] = 1$$

Assume scores are Gaussian

Closed-form expactation: single pass method!

Efficiency



Gaussian version is **~3x** faster than FlashDecoding!

Dash Attention

w/


Yuxiang Haung


Nuno Gonçalves


Federico Alvetreti


Lei Li


Xu Han

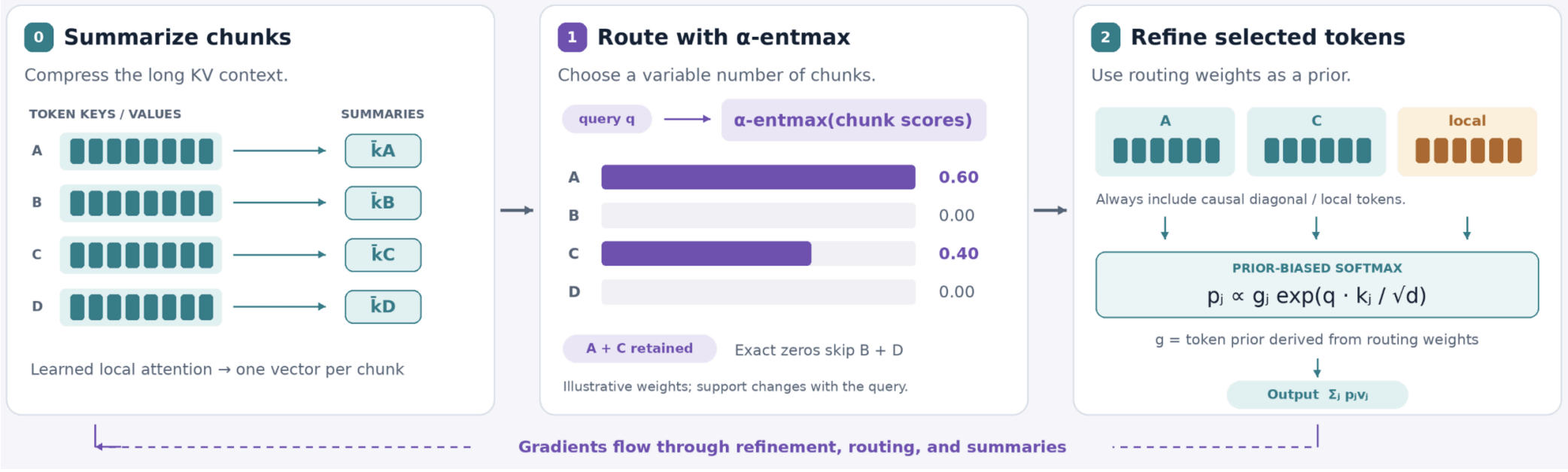

Edoardo Ponti


André Martins

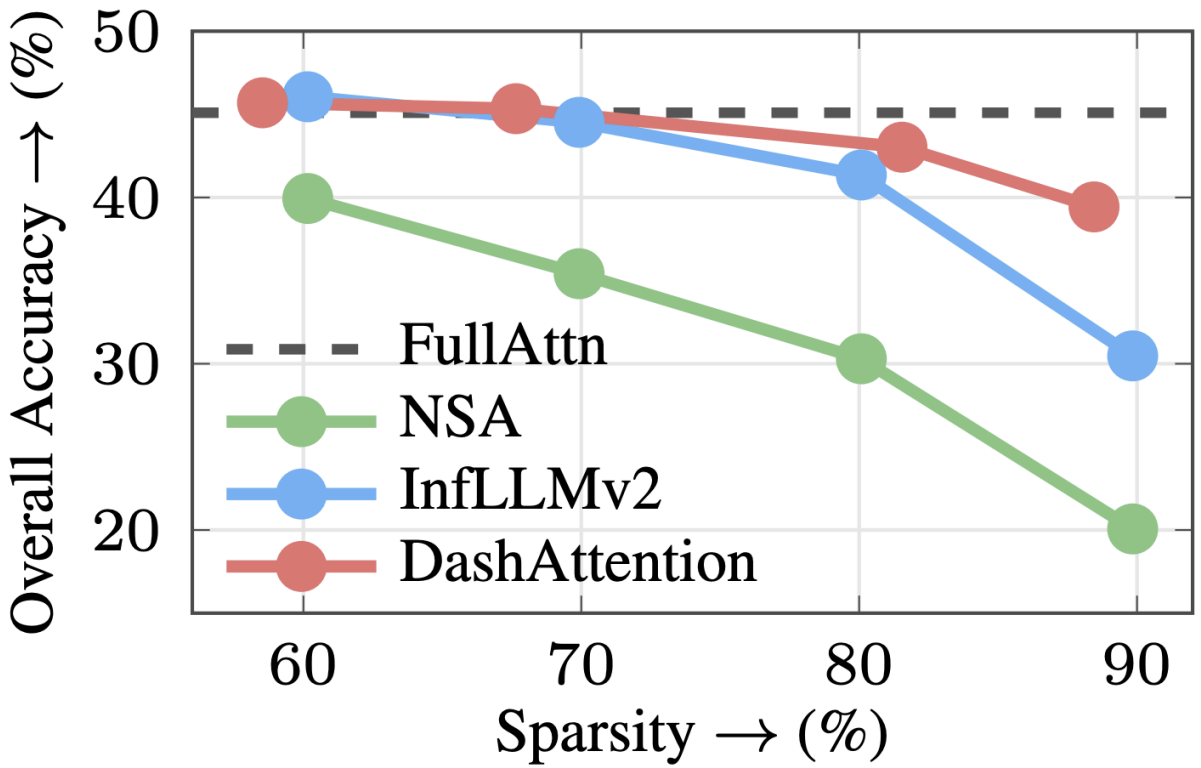
📌 NeurIPS 2026

📄 arxiv.org/abs/2605.18753

💻 github.com/fasa-org/dash-attention/



HELMET Benchmark (long-context)



Efficiency

	Prefill									Decoding								
	$s = 75\%$			$s = 87.5\%$			$s = 93.7\%$			$s = 75\%$			$s = 87.5\%$			$s = 93.7\%$		
N	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**

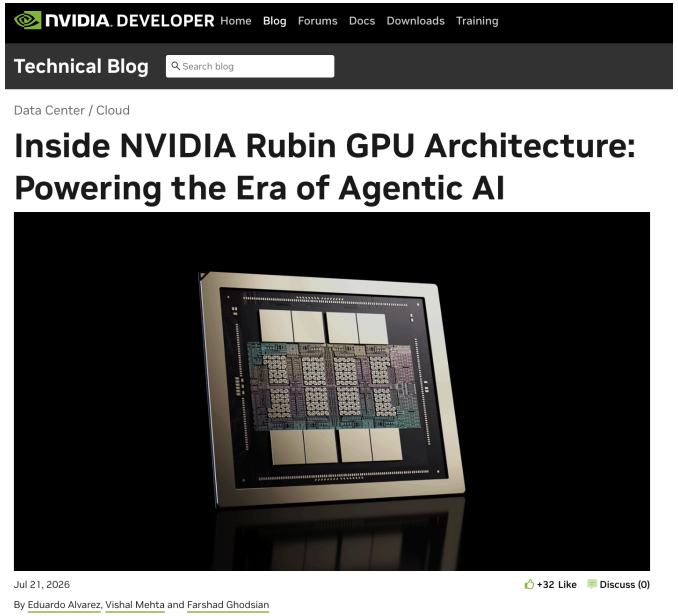
Efficiency

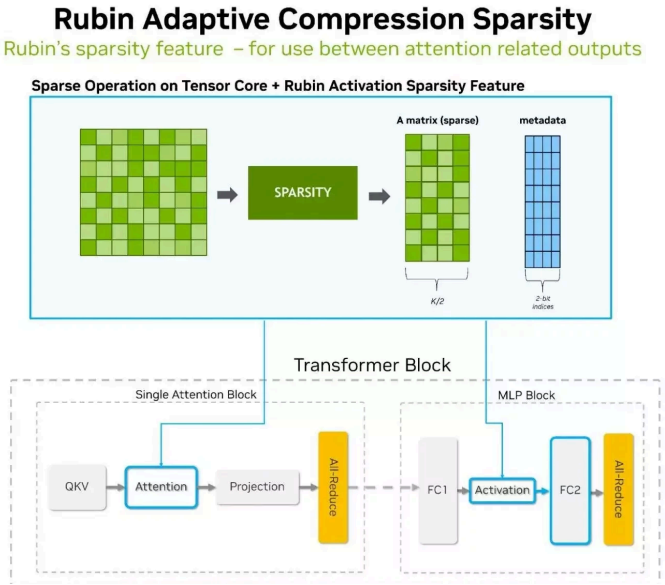
	Prefill									Decoding								
	$s = 75\%$			$s = 87.5\%$			$s = 93.7\%$			$s = 75\%$			$s = 87.5\%$			$s = 93.7\%$		
N	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA	NSA	Iv2	DA
16K	0.71	0.96	**1.34**	1.23	1.63	**2.09**	1.91	2.40	**2.87**	0.40	1.20	**1.54**	0.43	1.43	**1.67**	0.46	1.73	**1.93**
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 FlashAttention on small sequences, *faster and stronger* on **long sequences**.





GPUs + Dynamic Sparsity = 🥰

What's next? 🫣

Final Thoughts

Scaling sparse attention with next generation of EuroLLM --- Fully open models/data/code

EuroLLM
1.7B, 9B, 22B
1M+ downloads



EuroBERT
210M, 610M, 2.1B
300K+ downloads

Final Thoughts

Scaling sparse attention with next generation of EuroLLM --- Fully open models/data/code

EuroLLM
1.7B, 9B, 22B
1M+ downloads



EuroBERT
210M, 610M, 2.1B
300K+ downloads



SMURF4EU

COMPUTE ALLOCATION

1.9M GPU-hours in Jupiter

Extreme-Scale EuroHPC grant 🙌

CORE CAPABILITIES

Multilingual • Multimodal • Reasoning • Long-Context

Covering text, code, speech, vision, and video

LANGUAGE COVERAGE

All 24 official EU languages + 11 additional languages

Portuguese is included 🇵🇹 🇧🇷 🇦🇴



Beyond the lab



Gandara AI

Stay tuned, and let's talk.

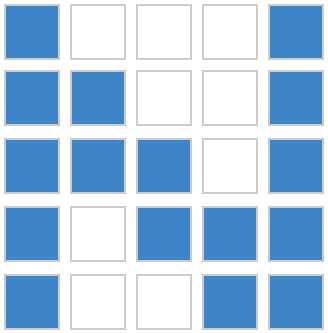
THANK YOU FOR YOUR ATTENTION



Want to collaborate? Drop me an email

Sparse Attention

Fixed Patterns



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

😕 **Low Adaptivity**

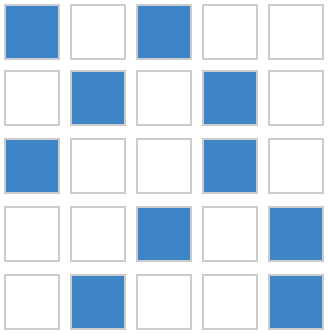
😕 **Low Flexibility**

😊 **Full Differentiability**

🚀 $O(n \times k)$ time cost

$O(n \times k)$ memory cost

Dynamic Selection



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

😊 **High Adaptivity**

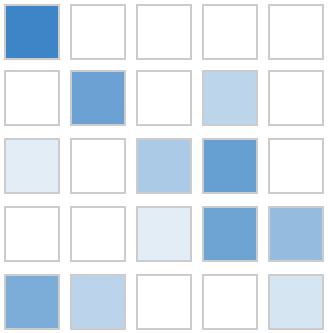
😐 **Medium Flexibility**

😕 **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

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

$$4096^2 \times 16 \times 32/8 = \textbf{1GB}$$ *+ model params and optimizer states*

for a single sample

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

$4096^2 \times 16 \times 32/8$ = **1GB** *+ model params and optimizer states*

for a single sample
with just 4K tokens!

⚡AdaSplash

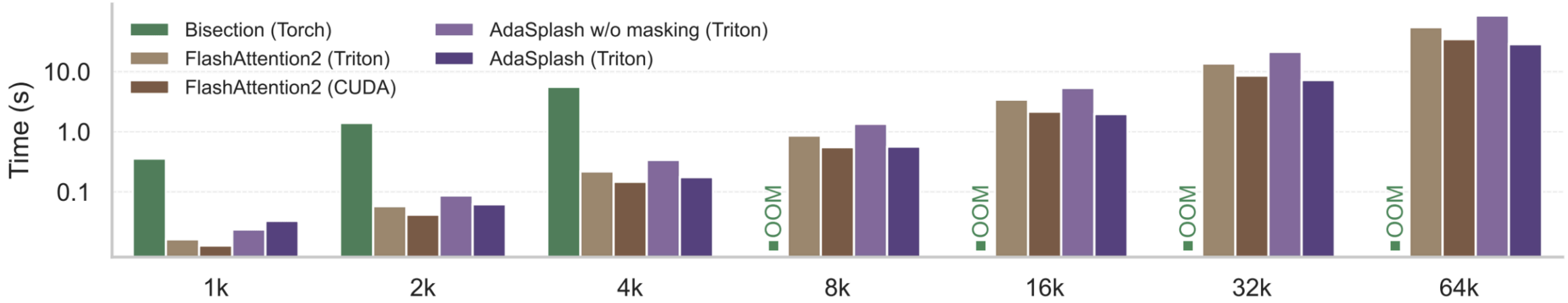
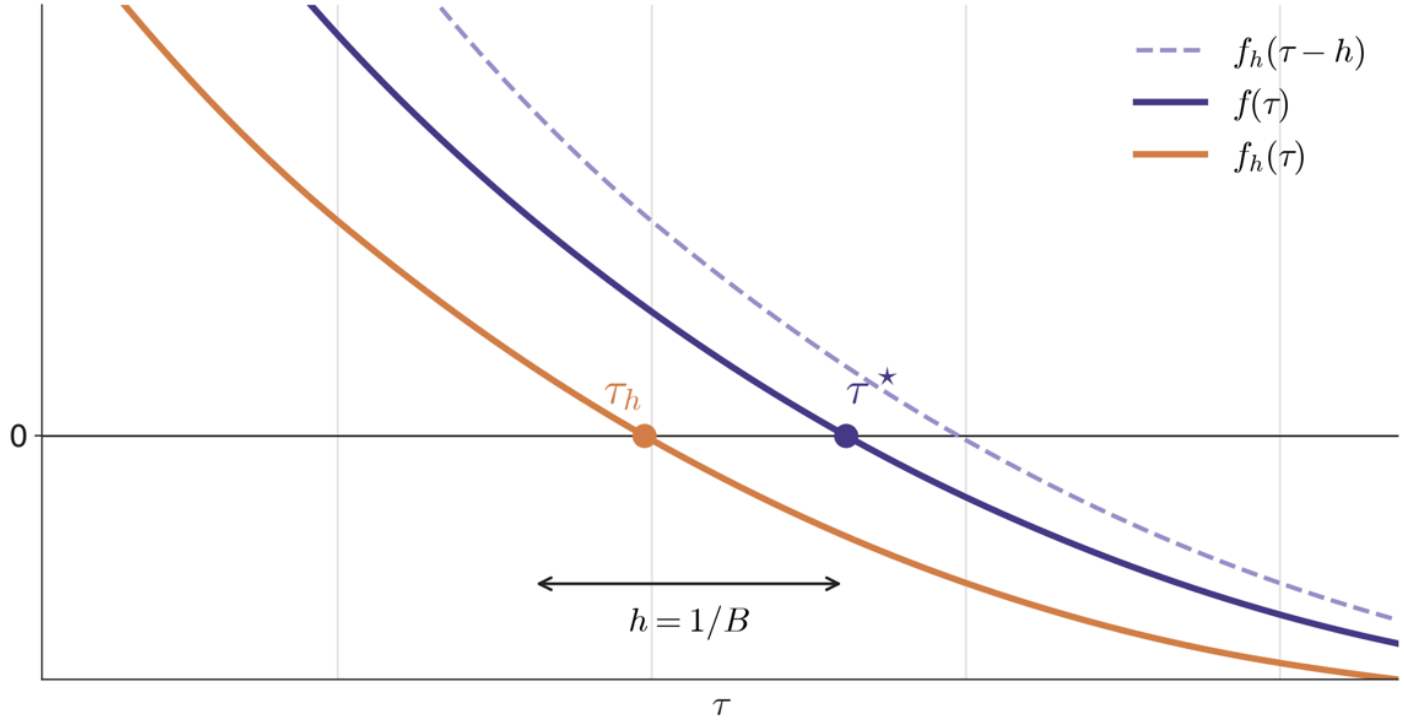


Figure 3. Efficiency of algorithms for computing non-causal attention in terms of the average training step time for increasingly longer sequence lengths. We use $\alpha = 1.5$ for α-entmax based methods (Bisection and ADASPLASH).

- Before AdaSplash, entmax attention (Bisection) was not scalable!
- AdaSplash, in Triton, can be bit faster than FlashAttention in CUDA!

⚡⚡ AdaSplash-2: Faster Differentiable Sparse Attention

$$f_h(\tau) \leq f(\tau) < f_h(\tau - h) \quad \forall \tau \in [0, 1] \qquad \Longrightarrow \qquad \tau^\star - h < \tau_h \leq \tau^\star$$



With $B = 8$ bins:

τ is known to within 1/8 of $\tau^\star$ before a single iteration!

Results

RULER (needle in a haystack) with 350M models

	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**

Results

RULER (needle in a haystack) with 350M models

	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**

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

	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**

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

	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**

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**