Scanning
Selective
Structured
State
Space
Sequence
(Models)

…also known as ***Mamba*** *(get it snake makes "s" noise or something)*

Albert Gu and Tri Dao. "Mamba: Linear-Time Sequence Modeling with Selective State Spaces".
In: arXiv preprint arXiv:2312.00752 (2023)

UBC MLRG Summer 2024
Alan Milligan
alanmil@cs.ubc.ca

Everyone Loves Transformers (totally…)

Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …

☆ Save 99 Cite Cited by 123728 Related articles All 91 versions + Add to Paperlib

Yearly citation count update: ≈**125,000**

5

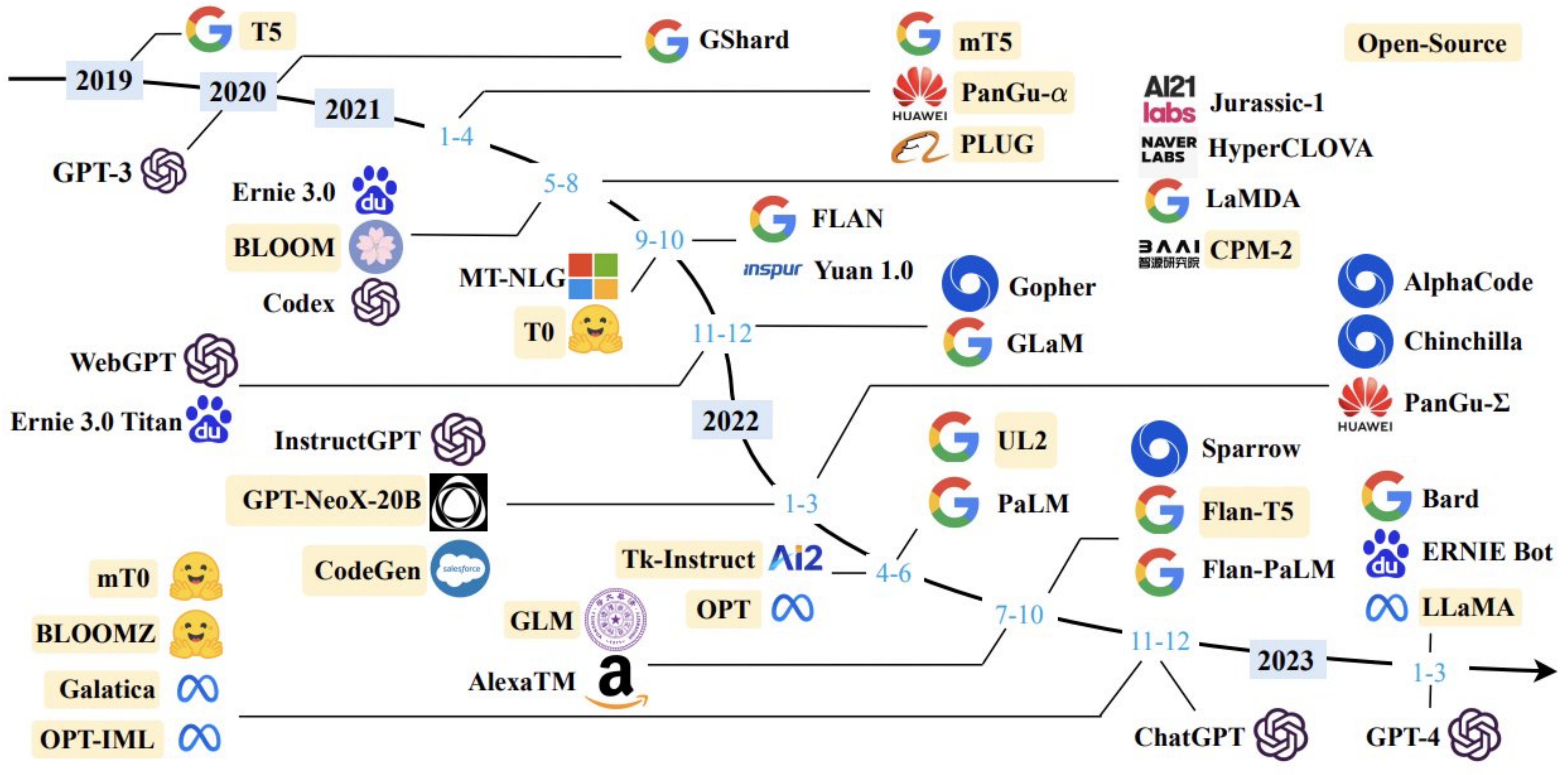


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

…but they are expensive.

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

Table 4: Comparison of carbon emissions between BLOOM and similar LLMs. Numbers in *italics* have been inferred based on data provided in the papers describing the models.

https://arxiv.org/pdf/2211.02001

CO_2 emissions are comparable to several international flights (per run)

- $2.5k - $50k (110 million parameter model)
- $10k - $200k (340 million parameter model)
- $80k - $1.6m (1.5 billion parameter model)

https://arxiv.org/pdf/2004.08900

Financial Costs are insane

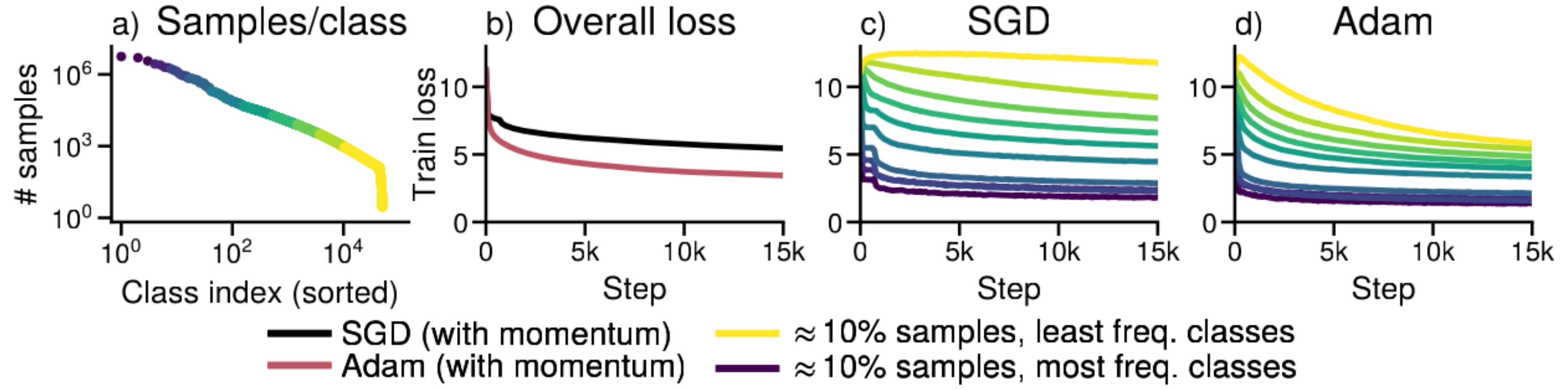


Figure 1: **Gradient descent does not make progress on low-frequency classes, while Adam does.** Training GPT2-Small on WikiText-103. **(a)** Distribution of the classes sorted by class frequency, split into groups corresponding to ≈10% of the data. **(b)** Overall training loss. **(c, d)** Training loss for each group using SGD and Adam. SGD makes little to no progress on low-frequency classes while Adam makes progress on all groups. **(b)** is the average of **(c, d)** for the respective optimizer.

Reality check: Using Lambda (cloud) this plot would have cost Fred and I ≈$18,000 USD

(Closer to $50k on AWS….)

Why the costs? (aside from parameter counts starting with a B)

Attention has Quadratic Everything

$$\text{Softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

Training

Let ℓ be the sequence length we train with

$$X \in \mathbb{R}^{\ell \times d}$$

$$QK^\top = W_Q X X^\top W_K \in \mathbb{R}^{\ell \times \ell}$$

Model	**Context length**
GPT 3.5	4,096
GPT 4	8,192
GPT 4-32k	32,768
Llama 1	2,048
Llama 2	4,096

$\mathcal{O}(\ell^2)$ FLOPs and memory is no fun

Flash Attention can get you down to $\mathcal{O}(\ell)$ memory but not FLOPs

Inference

Let ℓ be the sequence length predict

For the ℓth token, we do

$$\text{Softmax}\left(\frac{1}{\sqrt{d}}\begin{bmatrix}\mathbf{q}_\ell{}^\top\mathbf{k}_0 & \cdots & \mathbf{q}_\ell{}^\top\mathbf{k}_\ell\end{bmatrix}\right)V$$

Which is $\mathcal{O}(\ell)$ per token and so $\mathcal{O}(\ell^2)$ in total. No fun.

K and V don't need to be recomputed (KV Caching) but we can't get away from making that entire attention vector (especially since Softmax is non-linear)

Many attempts to address this

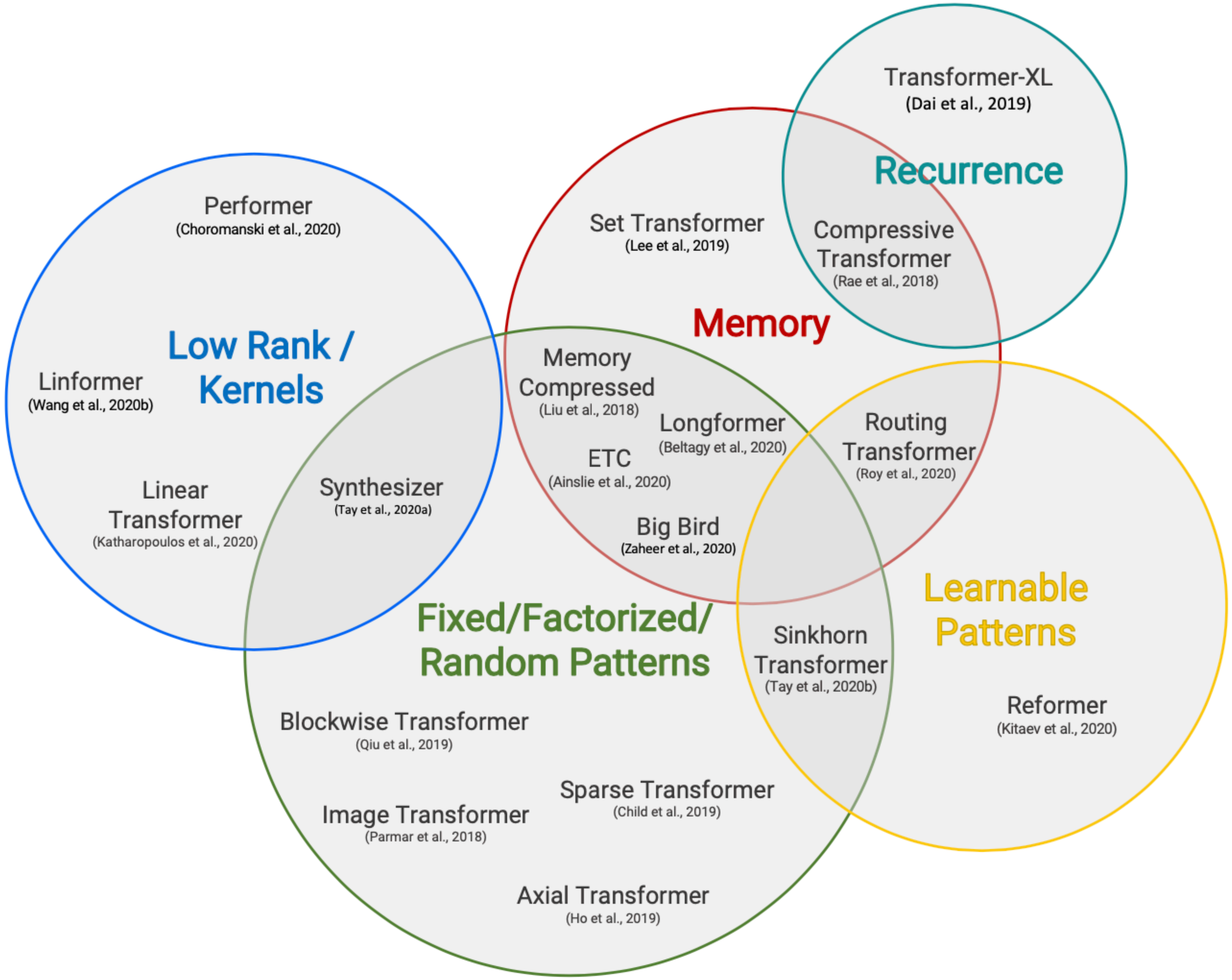


Figure 15.29: Venn diagram presenting the taxonomy of different efficient transformer architectures. From [Tay+20b]. Used with kind permission of Yi Tay.

Many attempted remedies: scary kernels, various linear approximations, sparse attention patterns, etc

None managed catch on in mainstream use cases

Why not do RNNs then?

RNN good?

- $\mathcal{O}(l)$ training step
- $\mathcal{O}(l)$ inference (constant per token)

RNN bad?

- fixed context size
- Unstable optimization
- **Can’t parallelize easily**

:(What do I do with the
350,000 H100s I just casually
purchased then
- zuck probably

It has been theorized if we could train larger RNNs, they would match transformers

PCMag

Zuckerberg's Meta Is Spending Billions to Buy 350000 Nvidia H100 GPUs

In total, Meta will have the compute power equivalent to 600000 Nvidia H100 GPUs to help it develop next-generation AI, says CEO Mark...

Q: Why can't RNNs parallelize well?

A: Nonlinear State Transitions

The activations σ in pretty much all RNNs is non-linear, so **must** compute all the h_t business before making h_{t+1}.

Basic RNN Structure

$$h_{t+1} = \sigma_h(W_h h_t + U_h x_t + b_h)$$
$$y_{t+1} = \sigma_o(W_o h_{t+1} + b_h)$$

What if we got rid of the nonlinear $\sigma_h(\cdot)$?

$$h_1 = U_h x_0$$
$$h_2 = W_h U_h x_0 + U_h x_1$$
$$h_3 = W_h^2 U_h x_0 + U_h x_1 + U_h x_2$$
$$\vdots = \vdots$$

(Simpler by assuming $h_0 = 0$ and ignoring biases)

This can be written as a convolution and is fast on hardware
(but let's switch to SSM notation first)

State Space Model Notation

State space models are **old** (control theory, bayesian stats, etc)

They are a way of modelling a system with input/output signals through time

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t)$$

$x =$ continuous time input signal

$h =$ continuous state (and its derivative)

$y =$ continuous time output signal

These are also called Linear Time Invariant models given the transition matrices don't depend on time

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t)$$

The $\mathbf{D}x(t)$ can be viewed as a residual connection so the papers involved leave it out of the math (but still implement it?)

Annoying detail: Discretization

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t)$$

These are functions of continuous time, but in things like next token prediction we have a discrete sequence of inputs

While I am personally not sure why this is necessary in a deep learning context, it is consistent with the theory of these models

$$h_{t+1} = \overline{\mathbf{A}}h_t + \overline{\mathbf{B}}x_t$$
$$y_t = \overline{\mathbf{C}}h_t$$

$$\overline{\mathbf{A}} = \exp(\Delta A)$$
$$\overline{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I})\Delta\mathbf{B}$$
$$\overline{\mathbf{C}} = \mathbf{C}$$

This discretization is called a *zero order hold* and Δ can be viewed as "how coarse" the discretization is (I don't have good intuition for this, and neither does anything I've read)

Back to RNN as a convolution

$$h_{t+1} = \overline{\mathbf{A}}h_t + \overline{\mathbf{B}}x_t$$
$$y_t = \overline{\mathbf{C}}h_t$$

If we expand this out like with the RNN, we get

$$\begin{aligned} y_1 &= \overline{\mathbf{C}}\overline{\mathbf{B}}x_1 \\ y_2 &= \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}x_1 + \overline{\mathbf{C}}\overline{\mathbf{B}}x_2 \\ y_3 &= \overline{\mathbf{C}}\overline{\mathbf{A}}^2\overline{\mathbf{B}}x_1 + \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}x_2 + \overline{\mathbf{C}}\overline{\mathbf{B}}x_3 \\ \vdots &= \vdots \end{aligned}$$

Then, we can do the following convolution fast on hardware (FFT and such)

$$\overline{\mathbf{K}} = (\overline{\mathbf{C}}\overline{\mathbf{B}}, \overline{\mathbf{C}}\overline{\mathbf{A}}\overline{\mathbf{B}}, \ldots, \overline{\mathbf{C}}\overline{\mathbf{A}}^{\ell}\overline{\mathbf{B}})$$
$$\mathbf{y} = \mathbf{x} * \overline{\mathbf{K}}$$

(where $*$ is the convolution operation)

Authors say this speedup allows you use 10-100 times larger hidden state than RNNs because smart implementations never have to materialize h_t

Matrix powers are scary

Great so now we can do fast sequence to sequence training, but $\mathbf{A}^{\ell}$ can be a big problem

Recall for general matrix $\mathbf{A}$,

$$\mathbf{A}^{\text{huge}} \to \begin{cases} 0 & \sigma_{\text{max}}(\mathbf{A}) < 1 \\ \infty & \sigma_{\text{max}}(\mathbf{A}) > 1 \end{cases}$$

Structured State Space Models

We have to get smart about parameterizing $\mathbf{A}$. Step 1 just make it diagonal. Step 2 cite this paper also by the authors that argues their initialization doesn't explode (too much linear algebra for slides)

Theorem 2. *The continuous- (3) and discrete- (4) time dynamics for **HiPPO-LegS** are:*

$$\frac{d}{dt}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \qquad (3)$$

$$c_{k+1} = \left(1 - \frac{A}{k}\right)c_k + \frac{1}{k}Bf_k \qquad (4)$$

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}, \qquad B_n = (2n+1)^{\frac{1}{2}}$$

Proposition 5. *For any times $t_0 < t_1$, the gradient norm of HiPPO-LegS operator for the output at time t_1 with respect to input at time t_0 is $\left\| \frac{\partial c(t_1)}{\partial f(t_0)} \right\| = \Theta(1/t_1)$.*

```
@article{hippo,
  title={HiPPO: Recurrent Memory with Optimal Polynomial Projections},
  author={Albert Gu and Tri Dao and Stefano Ermon and Atri Rudra and Christopher R\'{e}},
  journal={arXiv preprint arXiv:2008.07669},
  year={2020}
}
```

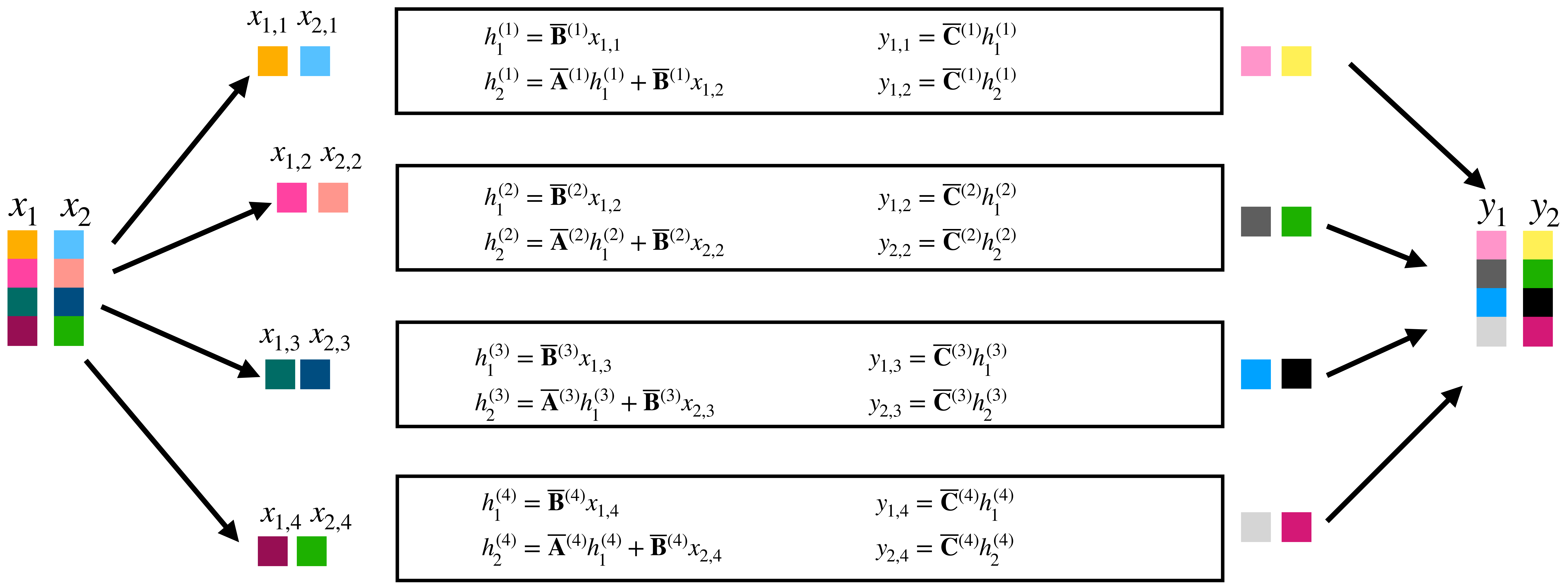
So it's just a diagonal matrix with $n+1$ on the nth diagonal at initialization. Also it's optimized in log space which is not mentioned in the paper

Digression #1: stack of scalar transforms

The $y(t)$ and $x(t)$ functions are typically considered $\mathbb{R} \to \mathbb{R}$. But token embeddings are in $\mathbb{R}^d$.

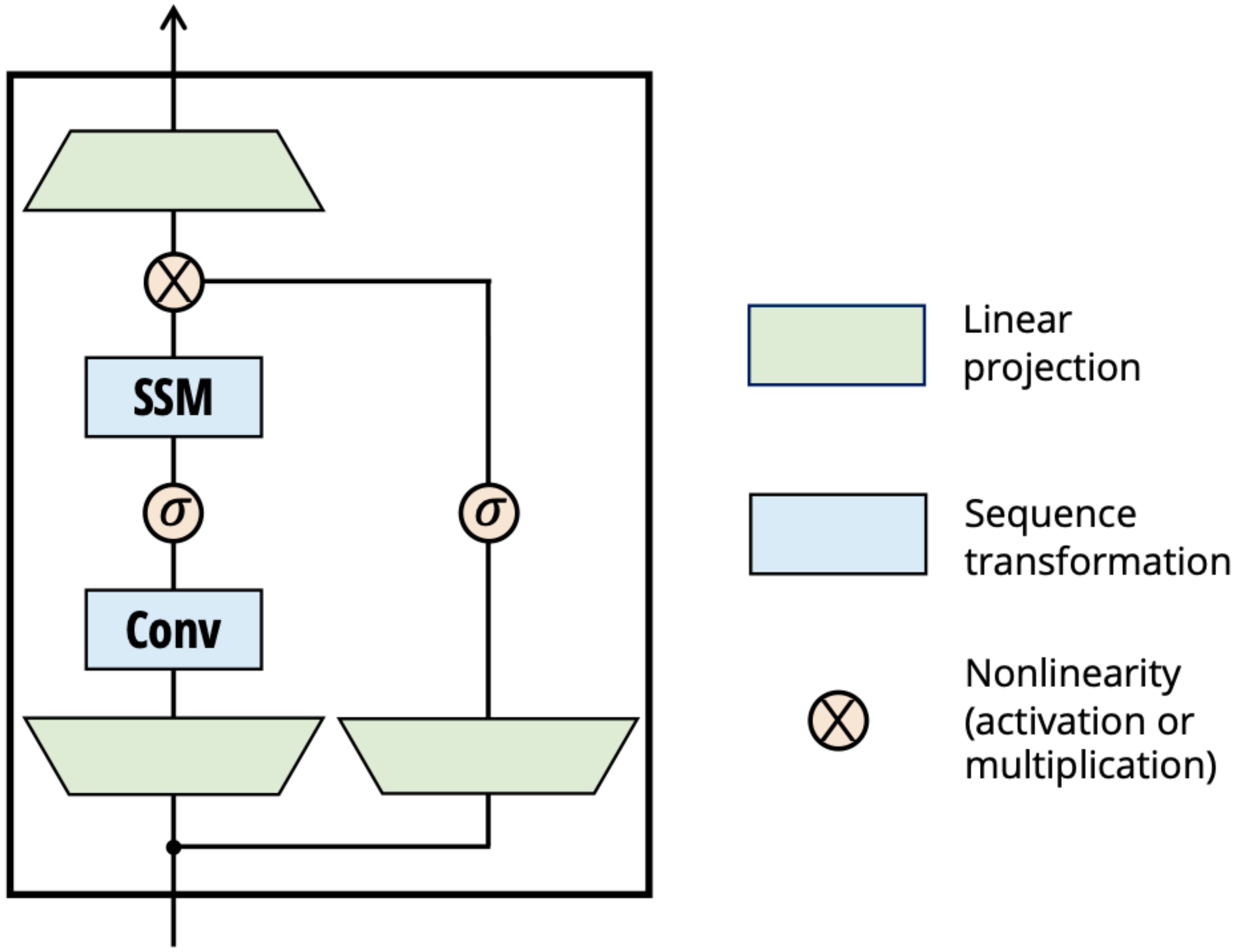
Instead of just making a vector valued SSM, they stack d independent univariate ones…

(the internal state of the SSM is vector valued though)



Digression #2: It's not just a big linear model

Mamba Block



All this SSM stuff is really just to replace the Attention Layer

The full model has plenty of deep learning flavour of the minute blocks including:

- Linear Layers
- Swish Activations
- 1D convolutions
- RMSNorm

Not actually Mamba

So far I have not actually described Mamba, I have described the "Structured State Space Sequence Model" (S4) while Mamba is (heavily) based on

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
4: $\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

B = Batch dimension
L = Sequence Length dimension
D = Embedding dimension
N = SSM State dimension

The parameter sizes listed are a bit misleading because the entries on the D dimension are use independently per channel, it's not like we ever do a (D,N) matmul

$\mathcal{O}(1)$ next token prediction in recurrent mode

Fast convolution for training

Problems with S4

Linear Time Invariance gets you fast training, but you can't treat inputs differently

$\mathbf{A}, \mathbf{B}, \mathbf{C}$ don't depend on x_t!

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$$
$$y(t) = \mathbf{C}h(t)$$

An analogy

Transformer	????	S4
"I don't need to remember what's important because I can look at every input for every prediction"	"I can decide what to remember based on what I think is important"	"I have to remember everything and I can't decide if some inputs are more important than others"

LSTMs would probably be in this bucket but they are slow and unstable

Although maybe not xLSTMs I have no idea

Adding an S: Selective

This is the key _algorithmic_ contribution: add input dependence for $\mathbf{B}$, $\mathbf{C}$ and Δ

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$

$\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow \mathsf{Parameter}$ ▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \mathsf{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \mathsf{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$ ▷ Time-varying: recurrence (*scan*) only
return y

As math,

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}(x(t))x(t)$$
$$y(t) = \mathbf{C}(x(t))h(t)$$

Or

$$h_{t+1} = \overline{\mathbf{A}}h_t + \overline{\mathbf{B}}(x_t)x_t$$
$$y_t = \overline{\mathbf{C}}(x_t)h_t$$

And as usual, $s_B(x)$, $s_C(x)$ and $s_\Delta(x)$ are neural networks

…but there's no nice fast convolutional form anymore which kind of defeats the purpose right?

A note on Δ

We can see that this discretization parameter is now a learned function of the input

$$\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_{\Delta}(\mathsf{Parameter} + s_{\Delta}(x))$$

Supposed $\Delta \to 0$

$$\overline{\mathbf{A}} = \exp(\Delta A) \to \mathbf{I}$$

$$\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - \mathbf{I})\Delta \mathbf{B} \to \mathbf{0}$$

$$h_{t+1} = \mathbf{I} h_t + \mathbf{0} x_t$$

This corresponds to ignoring the current input

Supposed $\Delta \to \infty$

Authors claim this means the current input overwrites the hidden state. I cannot figure out why that is the case mathematically unlike in the other cas

Assuming these claims are true, this connects the learned discretization to "gating" in RNNs such as LSTMs and GRUs

Final S: Scanning

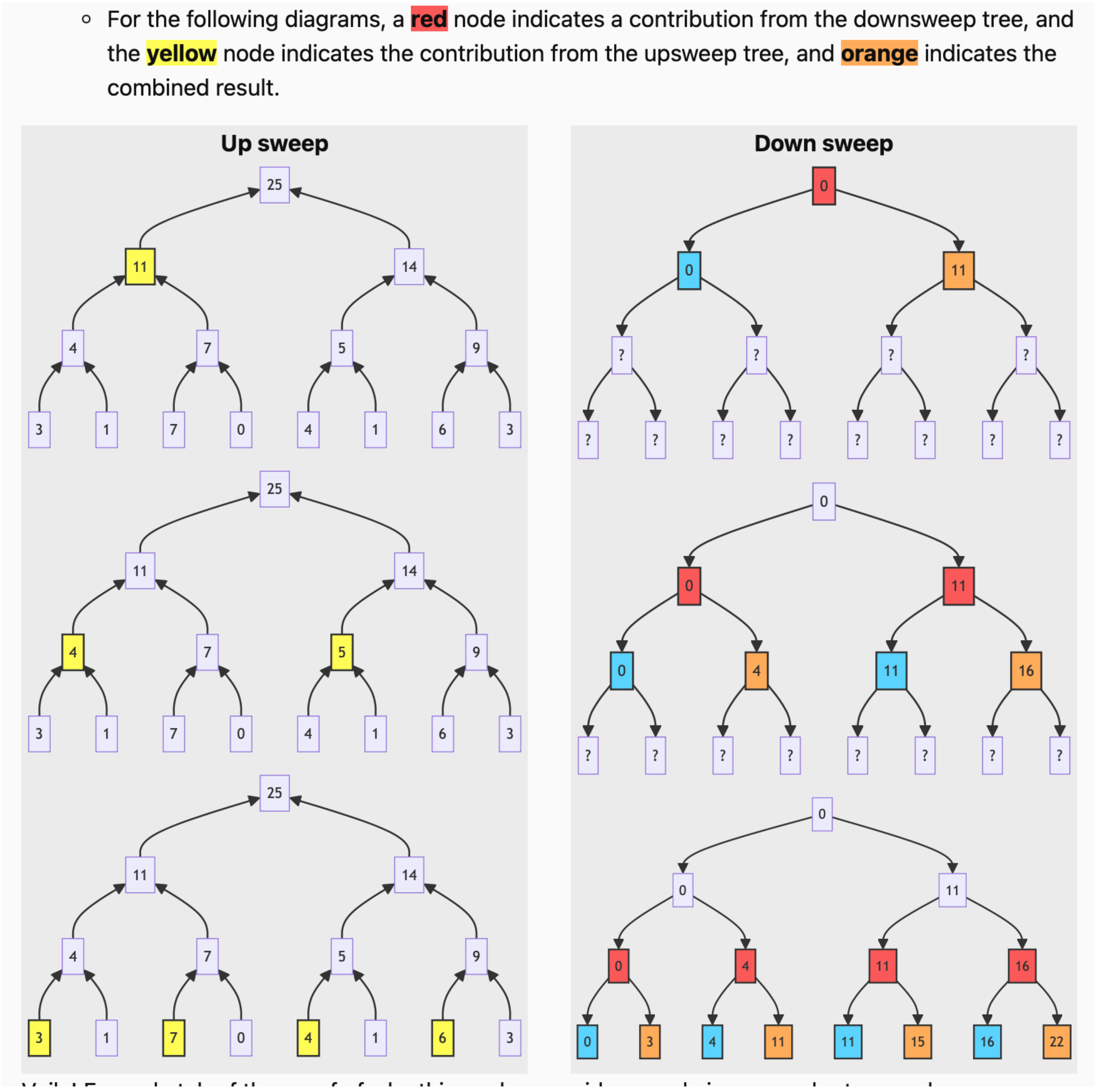
We have hit the limit of my knowledge for the second contribution.

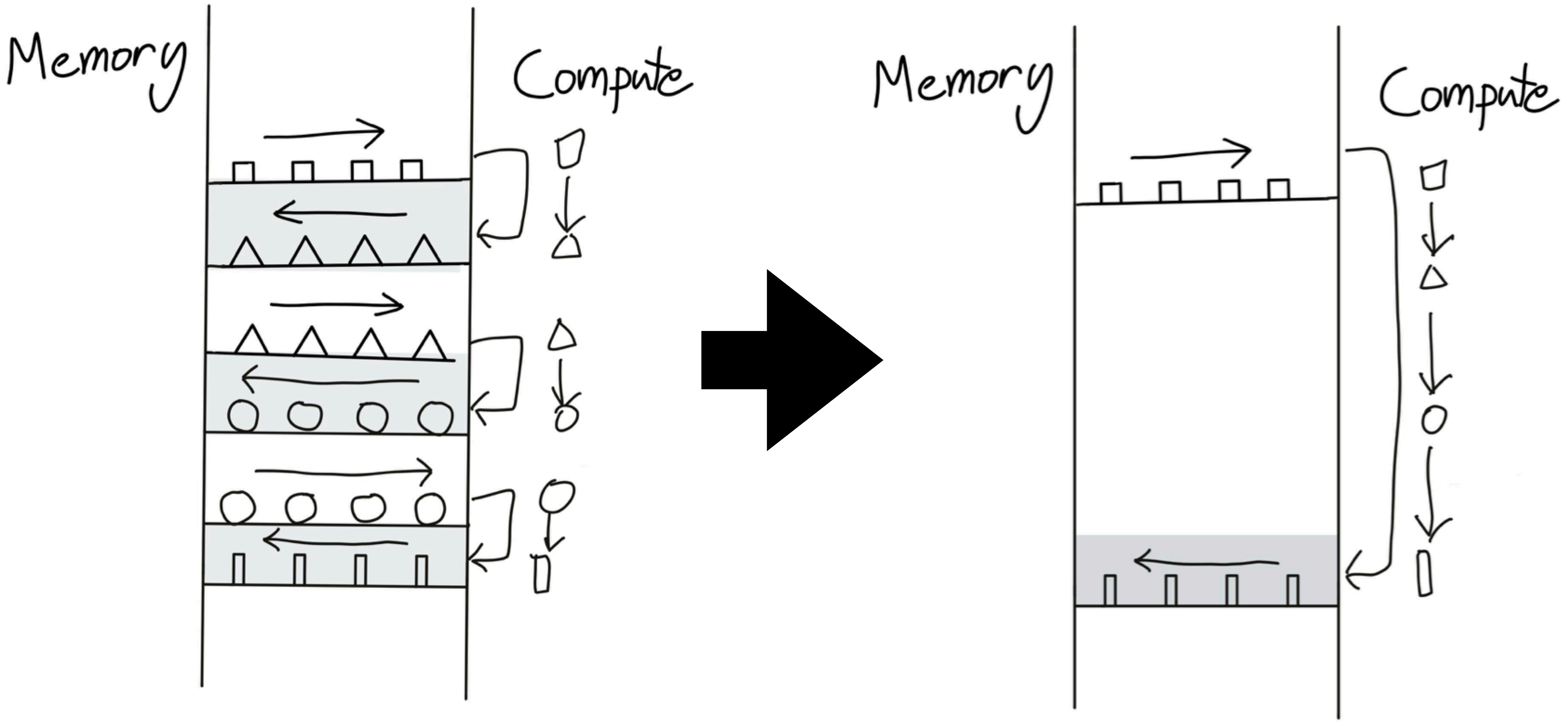
Even without the convolution, the authors figured out a way to make a fast on GPU algorithm.

Blelloch parallel prefix scan

Kernel Fusion

The SSM operation can be written as a prefix sum, naively $\mathcal{O}(\ell)$ but has a fast parallel algorithm





Typically, GPUs will load data into the fast memory, do something, and then write it back. If you have a chain of operations you can do in sequence, you can remove the back and forth writing (slow)

I did not have time to figure out this algorithm in detail but here are some resources

https://jameschen.io/jekyll/update/2024/02/12/mamba.html#the-blelloch-parallel-prefix-scan
https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda

Model Summary

SSMs	Selective	Scanning
Linear RNN with fast training and constant time inference	Linear RNN where the model parameters are a function of the inputs	Smart algorithm to be fast on hardware

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)
RNNs	**Slow...** (not parallelizable)	**Fast!** (scales **linearly** with sequence length)
Mamba	**Fast!** (parallelizable)	**Fast!** (scales **linearly** with sequence length + **unbounded** context)

Authors claim 5x inference and 3x training speedup and over transformers

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

So does it work? (generic results table)

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

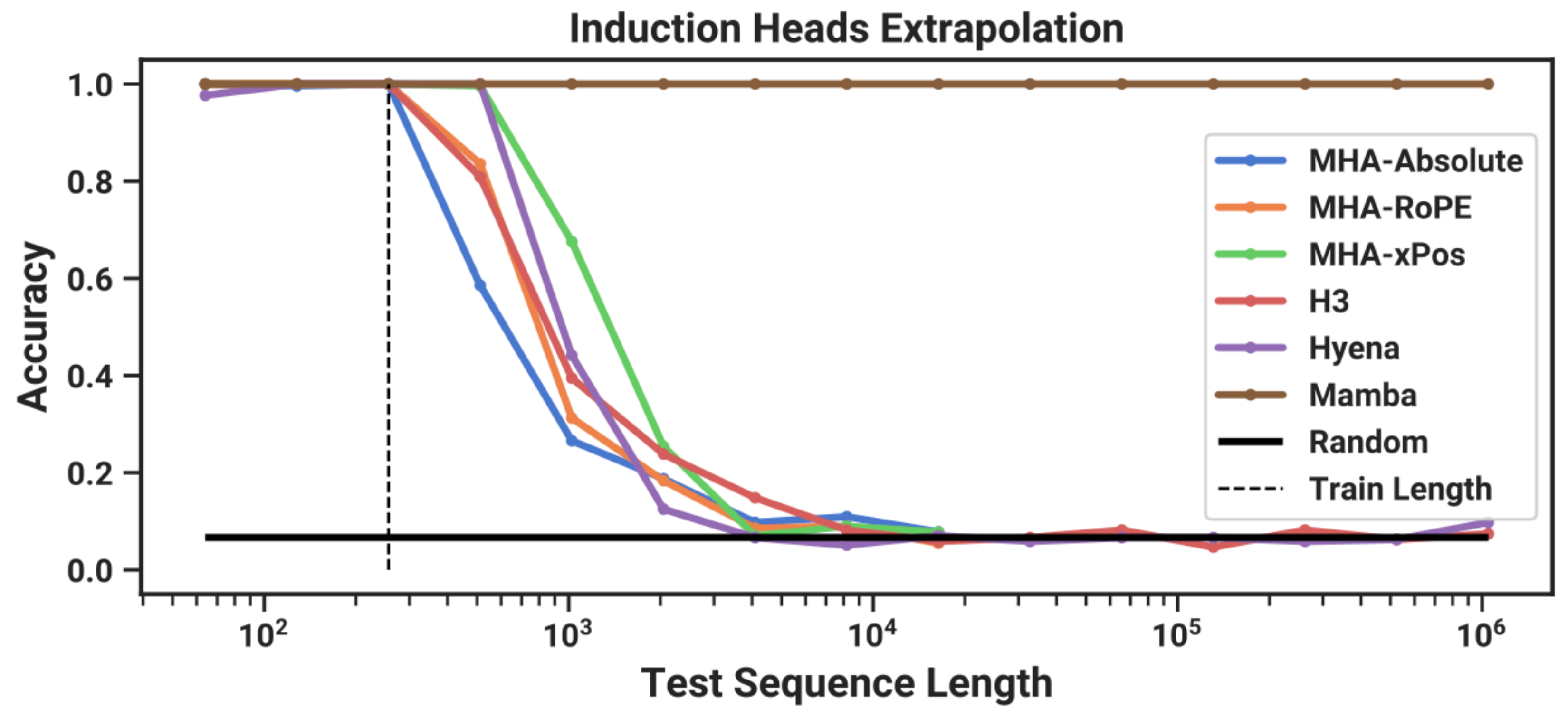
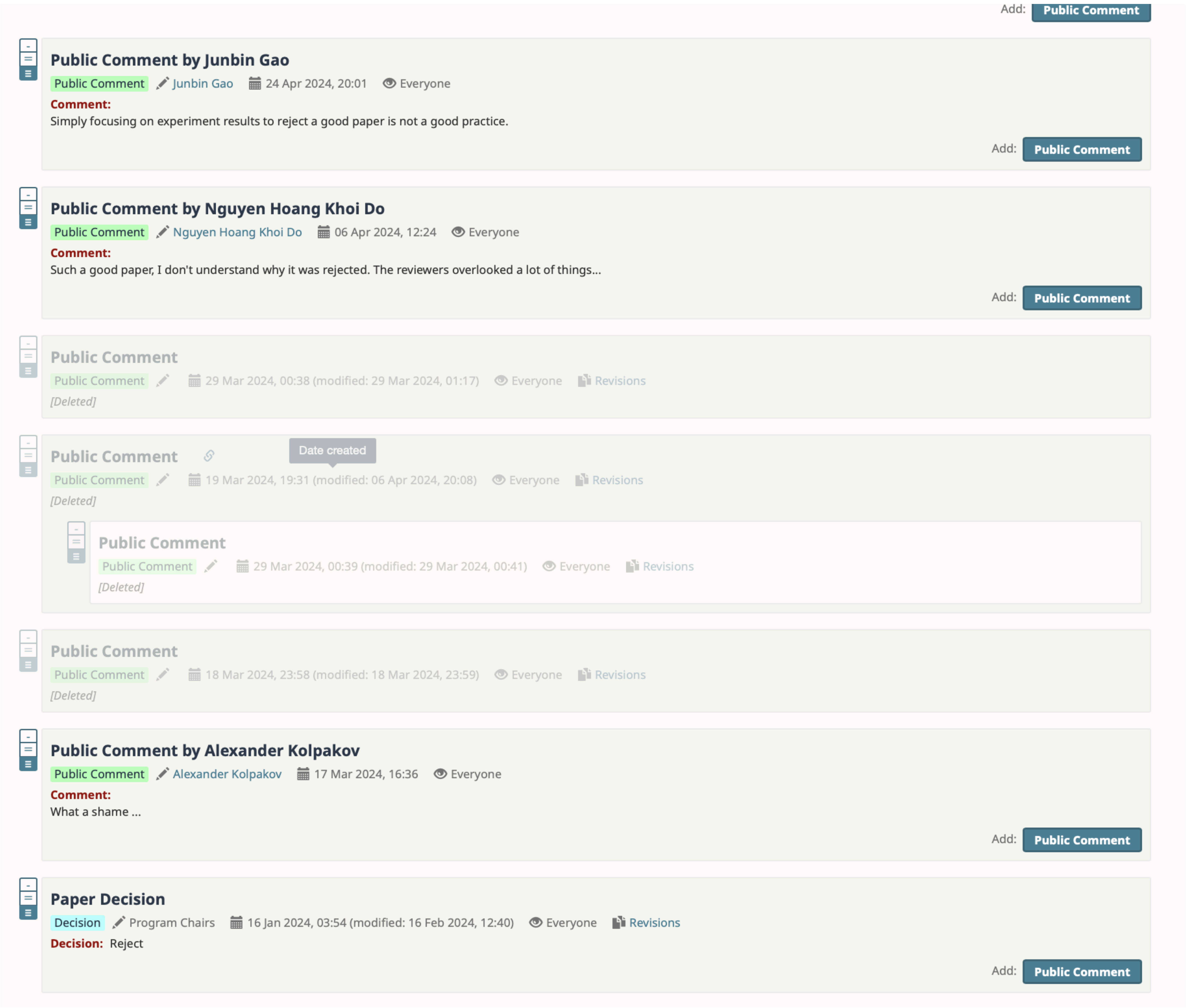


Table 2: (**Induction Heads.**) Models are trained on sequence length 2^8 = 256, and tested on increasing sequence lengths of $2^6 = 64$ up to 2^{20} = 1048576. Full numbers in Table 11.

Appears to perform well against models of similar size/larger

I am yet to see any company throw millions of dollars of compute at one of these

Still got rejected from ICLR tho


Add: Public Comment

Public Comment by Junbin Gao
Public Comment Junbin Gao 24 Apr 2024, 20:01 Everyone
Comment:
Simply focusing on experiment results to reject a good paper is not a good practice.
Add: Public Comment

Public Comment by Nguyen Hoang Khoi Do
Public Comment Nguyen Hoang Khoi Do 06 Apr 2024, 12:24 Everyone
Comment:
Such a good paper, I don't understand why it was rejected. The reviewers overlooked a lot of things...
Add: Public Comment

Public Comment
Public Comment 29 Mar 2024, 00:38 (modified: 29 Mar 2024, 01:17) Everyone Revisions
[Deleted]

Public Comment Date created
Public Comment 19 Mar 2024, 19:31 (modified: 06 Apr 2024, 20:08) Everyone Revisions
[Deleted]

Public Comment
Public Comment 29 Mar 2024, 00:39 (modified: 29 Mar 2024, 00:41) Everyone Revisions
[Deleted]

Public Comment
Public Comment 18 Mar 2024, 23:58 (modified: 18 Mar 2024, 23:59) Everyone Revisions
[Deleted]

Public Comment by Alexander Kolpakov
Public Comment Alexander Kolpakov 17 Mar 2024, 16:36 Everyone
Comment:
What a shame ...
Add: Public Comment

Paper Decision
Decision Program Chairs 16 Jan 2024, 03:54 (modified: 16 Feb 2024, 12:40) Everyone Revisions
Decision: Reject
Add: Public Comment

Thanks for listening!