

# Inventing Sequence Models as Vectorized Signal Processors

Julius Smith  
West Coast Machine Learning

June 20, 2024



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Abstract

Today's sequence models (such as large language models) in machine learning (AI) arose from a blend of principle-based design and empirical discovery, spanning several fields. This talk describes how the ideas could have emerged from an elementary signal-processing approach. This viewpoint offers some features:

1. Signal processing folks can quickly learn what is happening in a motivated way
2. Machine-learning experts might benefit from signal-processing insights
3. Obvious suggestions for things to try next naturally arise



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Background



# My Path to CCRMA (Center for Computer Research in Music and Acoustics)

Background

- [Path to CCRMA](#)

- [Courses](#)

Basic Idea

Architectures

Processing

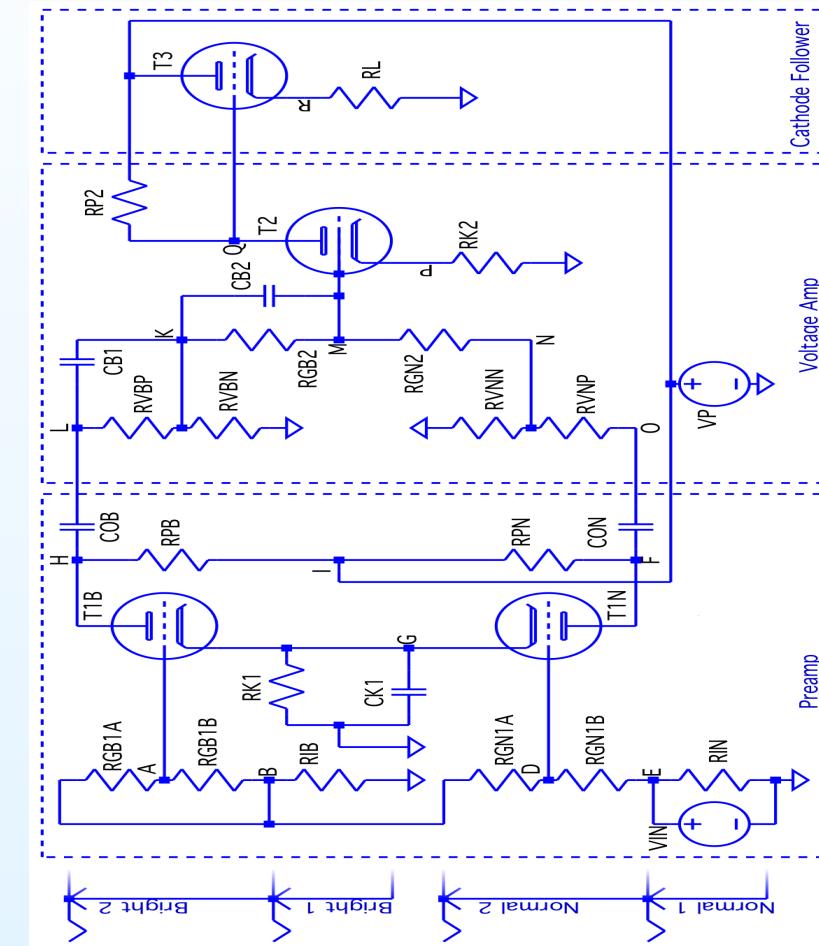
Attention

History Samples

Musician : Math : Physics : EE : Control : DSP : System ID : SAIL/CCRMA



(a) Some Gig



(b) Tube Amp



Background

• Path to CCRMA

• Courses

Basic Idea

Architectures

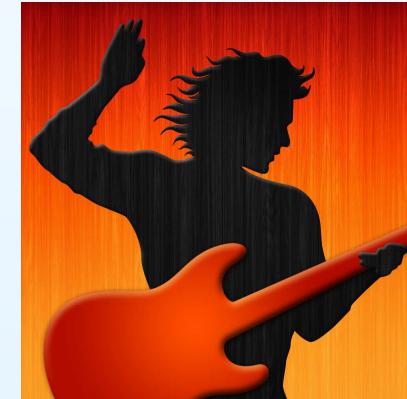
Processing

Attention

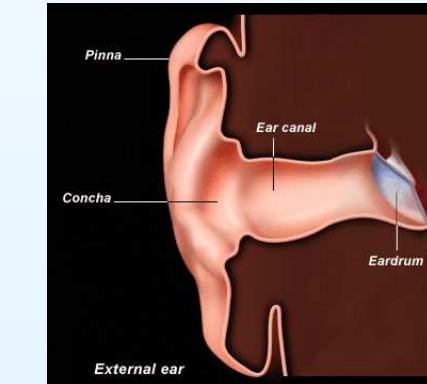
History Samples

## Courses Developed for CCRMA

- **Music 320A: AUDIO SPECTRUM ANALYSIS**
- **Music 320B: AUDIO FILTER ANALYSIS AND STRUCTURES**
- **Music 420A: PHYSICAL AUDIO SIGNAL PROCESSING**
- **Music 421A: TIME-FREQUENCY AUDIO SIGNAL PROCESSING**



420A



421A

All four textbooks **free online**





[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Music 320 (Signal Processing) Project Idea



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

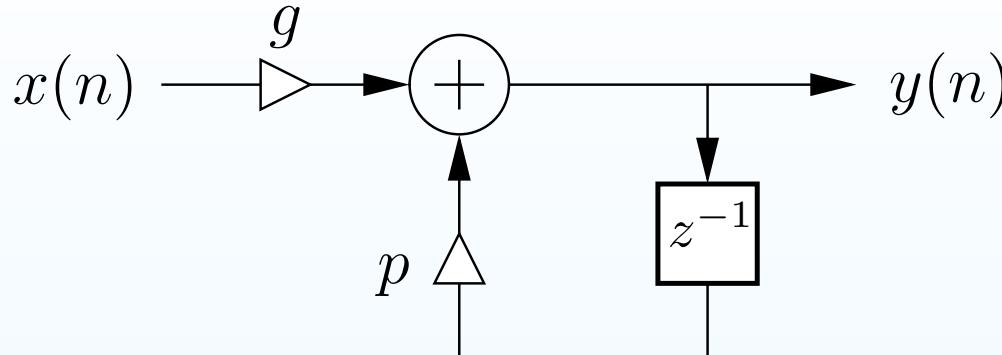
Architectures

Processing

Attention

History Samples

## Assignment: Do Something Cool with a *One-Pole Recursive Digital Filter*



One Pole at  $z = p$

$$y(n) = g x(n) + p y(n - 1), n = 0, 1, 2, \dots \quad [\text{difference equation}]$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{g}{1 - p z^{-1}} \quad [\text{transfer function}]$$

### Idea: Let's Make an Associative Memory!

- $x(n)$  can be a *long vector*  $\underline{x}(n) \in \mathbb{R}^N$  representing *anything we want* — any “label”
- Set  $g = 1$  and  $p = 1$  to make  $y(n)$  a *sum of all input vectors* (“integrator”)
- Choose the dimension  $N$  so large that *vectors in the sum are mostly orthogonal*
- Let  $\underline{x}(n)$  be **embedded vectors** (e.g., word2vec) so that *closeness = similarity*
- Retrieve similar vectors using a *matched inner product*  $\underline{w}^T \underline{x} > b$ ,  
for some suitable threshold  $b$  (Hey! That's a simulated neuron! (“Perceptron”))





Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Vector Retrieval by Inner Product

Given the sum of vectors

$$\underline{y}(n) = \sum_{m=0}^n \underline{x}(m)$$

and a “query vector”  $\underline{w} = \underline{x}(k)$ ,

find the query in the sum using an *inner product*:

$$\underline{w}^T \underline{y}(n) = \sum_{m=0}^n \underline{w}^T \underline{x}(m) \approx \underline{x}^T(k) \underline{x}(k) = \|\underline{x}(k)\|^2 > b(k)$$

where  $b(k)$  is the *detection threshold* for  $\underline{x}(k)$

- This works because the spatial dimension is so large that  $\underline{x}^T(j) \underline{x}(k) \approx \epsilon$  for  $j \neq k$
- Retrieval threshold  $b(k)$  depends on  $\|\underline{x}(k)\|^2$   
⇒ **suggestion:** reserve the radial dimension for similarity scoring
- I.e., only populate the **hypersphere** in  $\mathbb{R}^N$ :  $\|\underline{x}(k)\|^2 = 1$  (or  $N$ ),  $\forall k$
- We just invented RMSNorm, used extensively in neural networks (not LayerNorm)



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Decaying Vector Retrieval by Inner Product

RNNs typically have a *forgetting factor*  $p < 1$ .

Given the sum of vectors

$$\underline{y}(n) = \sum_{m=0}^n p^{n-m} \underline{x}(m)$$

and a “query vector”  $\underline{w} = \underline{x}(k)$ ,  
the inner product now gives

$$\underline{w}^T \underline{y}(n) = \sum_{m=0}^n \underline{w}^T p^{n-m} \underline{x}(m) \approx p^{n-k} \underline{x}^T(k) \underline{x}(k) = p^{n-k} > b$$

where  $b$  is the detection threshold for  $\underline{x}(k)$ , independent of  $k$  if  $\|\underline{x}(k)\| = 1$

- *Cannot retrieve when  $p^{n-k} < b$ , setting an upper limit on  $n$*
- We need  $p > b^{1/n}$  or  $n_{\max} \leq \log(b)/\log(p)$
- Lower  $b \Rightarrow$  more memory, but also more false detections from “interference” (“other vectors in the sum”)



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Orthogonality in High Dimensions

Let  $\mathbf{a} \in \mathbb{R}^N$  and  $\mathbf{b} \in \mathbb{R}^N$  be two normally random, real, unit-norm vectors in  $N$  dimensions, with  $\|\mathbf{a}\| = \|\mathbf{b}\| = 1$

The dot-product (inner product) of  $\mathbf{a}^T = [a_1, a_2, \dots, a_N]$  and  $\mathbf{b}^T = [b_1, b_2, \dots, b_N]$  is defined as

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \sum_{i=1}^N a_i b_i.$$

The squared dot product is

$$(\mathbf{a} \cdot \mathbf{b})^2 = \left( \sum_{i=1}^N a_i b_i \right)^2 = \sum_{i=1}^N \sum_{j=1}^N a_i a_j b_i b_j.$$

Expected value (average):

$$E[(\mathbf{a} \cdot \mathbf{b})^2] = \sum_{i=1}^N \sum_{j=1}^N E[a_i a_j] E[b_i b_j] = \sum_{i=1}^N \frac{1}{N} \frac{1}{N} = \boxed{\frac{1}{N}}$$



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Orthogonality in High Dimensions, Continued

We just showed the *expected squared dot product of two normally random unit vectors in  $\mathbb{R}^N$  is  $1/N$ , i.e.,*

$$E[(\mathbf{a} \cdot \mathbf{b})^2] = \frac{1}{N}$$

since  $E[a_i b_j] = 0$  for  $i \neq j$ ,  $E[a_i^2] = E[b_i]^2 = 1/N$ , and  $\mathbf{a}$  and  $\mathbf{b}$  are independent.

### Suggestions:

- *Initialize biases (detection thresholds) larger than  $1/N$*
- *Divide the sum of  $M$  vectors by  $\sqrt{M}$ :*
  - “power normalization”
  - “RMSNorm-preserving”
  - Done in Hawk & Griffin, e.g.
  - “Keep vector sums near the unit sphere”
- *Apply RMSNorm when training the initial vocabulary embedding (“word2sphere”)*
- *Set the model dimension just sufficient for the layer width at each level*
- **Caveat:** We are only considering one mechanism here — others are definitely going on



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Orthogonality of Random Sums

Similarly,

$$\begin{aligned} E \left[ \left( \underline{w}^T \underline{y}_n \right)^2 \right] &= E \left[ \left( \sum_{m=0}^n \underline{w}^T \underline{x}_m \right)^2 \right] = \sum_{l=0}^n \sum_{m=0}^n E \left[ \underline{w}^T \underline{x}_l \underline{x}_m^T \underline{w} \right] \\ &= \sum_{m=0}^n E \left[ \underline{w}^T \underline{x}_m \underline{x}_m^T \underline{w} \right] = \sum_{m=0}^n E \left[ \left( \underline{w}^T \underline{x}_m \right)^2 \right] = \boxed{\frac{n}{N}} \end{aligned}$$

assuming  $\underline{w} \notin \underline{y}$  and  $\|\underline{w}\| = \|\underline{x}_m\| = 1$  for all  $m$ . Thus, *retrieval becomes unreliable when the number of summed vectors  $n$  nears the model dimension  $N$ .*

- $N$  is of course the number of exactly orthogonal vectors possible in  $N$  dimensions
- If  $L$  vectors are typically in the sum, our Perceptron “bias” (detection threshold) should be higher than  $L/N$
- **Suggestion:** *Keep the number of active vectors in memory well below  $N$*



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## How Many Summed Vectors Needed for Language Parsing? (BERT style)

It is well known that *phone numbers* were limited to *7 digits* due to our *limited short-term memory* for *unrelated objects*. Can *language* be parsed using 7 vectors or less at each level? [Original Transformer paper had 8 attention heads and 6 layers (like neocortex)]

### Layers (e.g.):

1. Base vocabulary = characters  
(26 for English)
2. Syllable in 7 chars or less  
(44 syllables in English;  
107 in Int'l Phonetic Alphabet)
3. Word in 7 syllables or less
4. Noun + 6 or less modifying adjectives
5. Verb + up to 6 adverbs
6. Noun phrase
7. Direct or indirect object
8. Prepositional phrase
9. Subject, verb, [indirect object], object
10. Sentence
11. Paragraph
12. Section
13. Chapter
14. Book
15. Subject Area Hierarchy . . .

Different cortical areas (6 layers each) needed for this many levels.

### Complex Sentence Diagram Examples:

<https://www.quora.com/In-regards-to-diagramming-sentences-which-one-is-the-most-difficult-youve-ever-come-across>



[Background](#)

[Basic Idea](#)

- One Pole Filter
- Inner Product
- Orthogonality
- [Language](#)
- Model Dimension

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Pictorial Examples

- 6 dot-patterns on a die
- 7 LED segments for numbers
- 4 or fewer strokes to draw a letter

*Hierarchy* used to keep the number of components per concept *small*

### Suggestions:

- *Ready signal* when symbol is recognized (whole letter, word, phrase, etc.)
- *Reset memory* after symbol recognition
- Memory can be *small!*

Maybe these signals are happening in the layers already?



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Orthogonality of Exponentially Decaying Random Sums

RNNs typically have a *forgetting factor*  $p < 1$ , in which case we have, defining  $\mu = n - m$  and  $\lambda = n - l$ :

$$\begin{aligned} E \left[ \left( \underline{w}^T \underline{y}_n \right)^2 \right] &= E \left[ \left( \sum_{m=0}^n \underline{w}^T p^\mu \underline{x}_m \right)^2 \right] = \sum_{l=0}^n \sum_{m=0}^n E \left[ \underline{w}^T p^\lambda \underline{x}_l p^\mu \underline{x}_m^T \underline{w} \right] \\ &= \sum_{m=0}^n p^{2\mu} E \left[ \underline{w}^T \underline{x}_m \underline{x}_m^T \underline{w} \right] = \sum_{m=0}^n p^{2\mu} E \left[ \left( \underline{w}^T \underline{x}_m \right)^2 \right] \\ &= \boxed{\frac{1}{N} \frac{1 - p^{2(n+1)}}{1 - p^2}} \rightarrow \frac{1}{N} \frac{1}{1 - p^2} \quad (\text{as } n \rightarrow \infty) \end{aligned}$$

- For  $1/(1 - p^2) < N$ , keep  $p < \sqrt{(N - 1)/N}$
- For  $1/(1 - p^2) < N/2$ , keep  $p < \sqrt{(N - 2)/N}$

and so on. *This gives us one way to calculate a maximum feedback coefficient  $p$  in RNNs*



Background

Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

Architectures

Processing

Attention

History Samples

## Model Dimension vs. Model & Vocab Size (ChatGPT-4o, not checked)

- Original Transformer

- $N = 512$  (Model Size 65 M)

- BERT

- $N = 768$  (110 M, 30 K)
  - $N = 1024$  (340 M, 30 K)

- GPT-2

- $N = 768$  (124 M, 50 K)
  - $N = 1024$  (345 M, 50 K)
  - $N = 1280$  (774 M, 50 K)
  - $N = 1600$  (1.5 B, 50 K)

- GPT-3

- $N = 2048$  (2.7 B, 50 K)
  - $N = 4096$  (6.7 B, 50 K)
  - $N = 6144$  (13 B, 50 K)
  - $N = 12288$  (175 B, 50 K)

- T5

- $N = 512$  (60 M, 32 K)
  - $N = 768$  (220 M, 32 K)
  - $N = 1024$  (770 M, 32 K)
  - $N = 1024$  (3 B, 32 K)
  - $N = 1024$  (11 B, 32 K)

- ALBERT

- $N = 768$  (12 M, 30 K)
  - $N = 1024$  (18 M, 30 K)
  - $N = 2048$  (60 M, 30 K)
  - $N = 4096$  (235 M, 30 K)

- DistilBERT

- $N = 768$  (66 M, 30 K)

- Megatron-Turing NLG

- $N = 20480$  (530 B, 50 K)



## Background

### Basic Idea

- One Pole Filter
- Inner Product
- Orthogonality
- Language
- Model Dimension

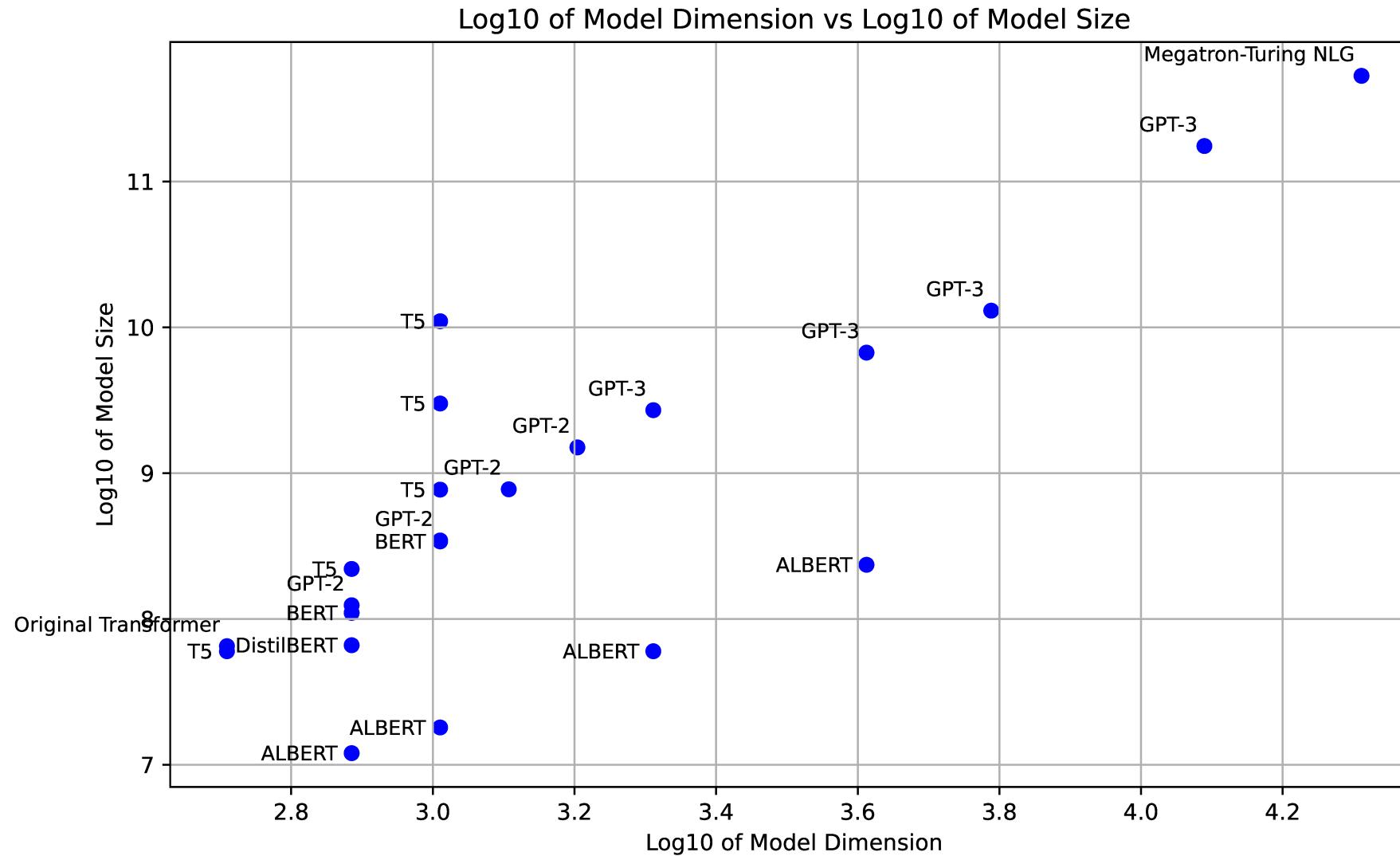
### Architectures

### Processing

### Attention

### History Samples

# Log10 Model Dimension versus Log10 Model Size





[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Architectures



[Background](#)

[Basic Idea](#)

[Architectures](#)

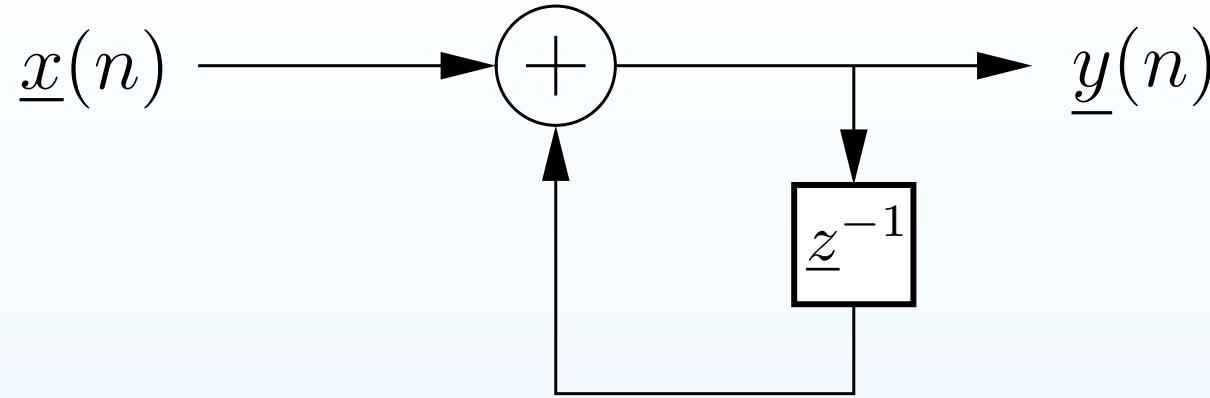
- [Vector Memory](#)
- Gating
- Gated RNN
- Skip Connection
- State Expansion

[Processing](#)

[Attention](#)

[History Samples](#)

## Cumulative Vector Memory



MidJourney



Background

Basic Idea

Architectures

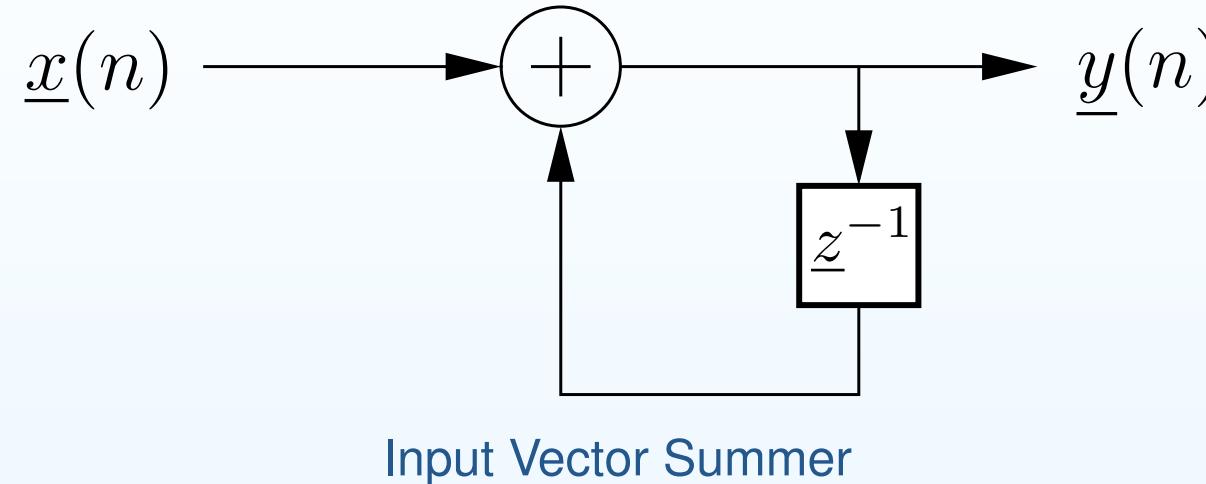
- Vector Memory
- Gating
- Gated RNN
- Skip Connection
- State Expansion

Processing

Attention

History Samples

## Gated Vector Memory



- **Problem:** Need a *memory reset*
- **Solution:** Set *feedback gain to zero* for one step to clear the memory
- **Problem:** Need an *input gate* to suppress unimportant inputs
- **Solution:** Set *input gain to zero* for unimportant inputs
- We just invented **gating**, used extensively in neural sequence models



Background

Basic Idea

Architectures

- Vector Memory
- Gating
- **Gated RNN**
- Skip Connection
- State Expansion

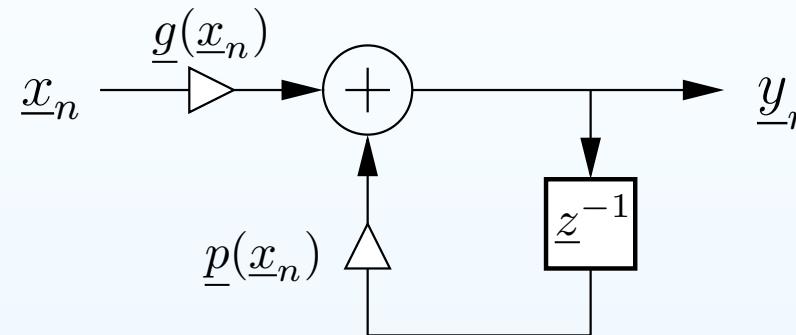
Processing

Attention

History Samples

## Gated Recurrent Network

**Idea:** Learn the input and feedback gates as functions of input  $\underline{x}_n$  based on many input-output examples  $(\underline{x}_n, \underline{y}_n)$  (“training data”):



Vector Memory with Learned Input and Feedback Gates

### Suggestions:

- Use learned, input-based, *activations* for gating (LSTM, GRU, Mamba smoothed input)
- While activated, *optionally* set *memory duration* via  $\underline{p}$  magnitude (SSMs, Mamba)
  - Initialize  $\underline{p}$  for desired initial memory duration (exponential fade time)
  - Learn  $\underline{p}(\underline{x}_n)$  as  $\mathbf{I} \cdot e^{-\Delta} \approx \mathbf{I} - \mathbf{I}\Delta$ , where  $\Delta = \text{softPlus}(\text{parameter}(\underline{x}_n, \underline{y}_n))$  (guaranteed stable — no “exploding gradients”) [Also multiply  $\underline{g}(\underline{x}_n)$  by  $\Delta$ ]
  - Consider *separate meaning-driven activation* multiplying feedback:  $\sigma(\mathbf{L}\underline{x})\underline{p}(\underline{x})$



Background

Basic Idea

Architectures

- Vector Memory
- Gating
- Gated RNN
- Skip Connection
- State Expansion

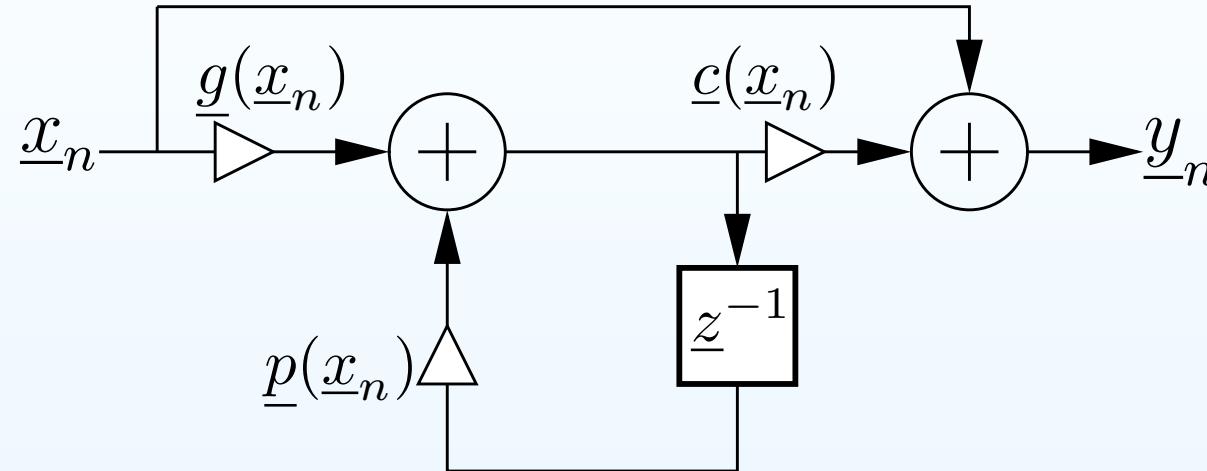
Processing

Attention

History Samples

## Output Gating

Idea: Since we have input and feedback gates, why not an *output gate* and bypass?



Gated RNN with Skip Connection

Output gating allows network to be “bypassed” when not helpful.

- **“Obvious” Suggestion:** The bypass path should be scaled for *power normalization*
- **Better yet:** Don’t scale the bypass and use RMSNorm at the input of the next layer (prevents a “bad layer” from isolating deeper layers from the input with garbage)



Background

Basic Idea

Architectures

- Vector Memory
- Gating
- Gated RNN
- Skip Connection
- State Expansion

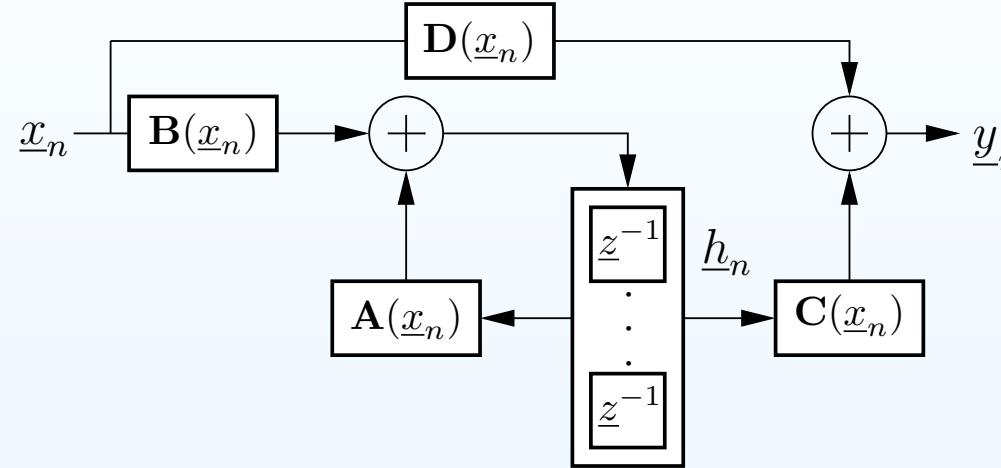
Processing

Attention

History Samples

## State Expansion

Idea: *Expand vector-memory dimension to an integer multiple of the model dimension:*



“*Structured State-Space Models*” (SSM) look like this (e.g., Mamba)

- Increased storage capacity (more vectors can be summed and later retrieved)
- Feedback matrix  $\mathbf{A}$  typically *diagonal* since 2022 (see “S4D”)  
⇒ Parallel bank of vector one-poles (“*linearly*” gated, *state-expanded RNNs*)
- In Mamba-2,  $\mathbf{A} = p \mathbf{I}$ , i.e., *shared memory duration* across expanded state
- Gating matrices in Mamba[-2] are simple linear input projections:  
$$[\mathbf{B}(\underline{x}_n), \mathbf{C}(\underline{x}_n)] = \mathbf{L} \underline{x}_n$$



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Memory Access



Background

Basic Idea

Architectures

Processing

- Perceptrons

- Sequences

- WRoPE

- Compressed Time

- Reservations

Attention

History Samples

## Detecting Multiple Vectors in Parallel

**Idea:** Detect multiple memory vectors in *parallel* using an array of “*Perceptrons*”

- Each Perceptron detects one or more memory vectors similar to its weight vectors

$$y_i(n) = \underline{w}_i^T \underline{h}(n) > b_i$$

where  $y_i(n)$  denotes the  $i$ th output at time  $n$ ,  $i=0,\dots,M-1$ ,  
and  $\underline{h}(n)$  denotes the (“hidden” [expanded-] state) vector memory

- Note that the  $\mathbf{C}$  matrix can provide these weights:

$$\underline{y}(n) = \mathbf{C} \underline{h}(n) > \underline{b}$$

- The Perceptrons indicate *which weight-vectors  $\underline{w}_i$  are present* in the vector memory  $\underline{h}$
- **Idea:** (Backpropagation—1980s?): To facilitate *learning  $\underline{w}_i$*  via gradient descent,  
replace “ $>$ ” by something smoother, such as  $1 + \tanh[\mathbf{C} \underline{h}(n) - \underline{b}]$



Background

Basic Idea

Architectures

Processing

• Perceptrons

• Sequences

• WRoPE

• Compressed Time

• Reservations

Attention

History Samples

## Sequence Modeling

- If each vector represents a *word*, a vector sum is simply a *bag of words*
- To model a *sequence* of words, we have various *sequence-position-encoding* options:
  1. *Amplitude Decay* - Multiply the sum by a *forgetting factor* each sequence step (RNNs) - *poor choice* (conflates with angular distance on the hypersphere)
  2. *Sinusoidal Amplitude Modulation* - Add a sinusoid with *increasing frequency* to each vector summing into the history (used in the original Transformer)
  3. *Phase Shift* - Multiply by the sum by  $e^{j\Delta}$  each sample (“RoPE”) - *apparently most used today*



Background

Basic Idea

Architectures

Processing

• Perceptrons

• Sequences

• WRoPE

• Compressed Time

• Reservations

Attention

History Samples

## RoPE and WRoPE

- Rotational Positional Encoding (RoPE) owns *one arc direction* along the hypersphere
- We can thus rotate our vector memory  $\underline{h}(n)$  by  $\Delta$  radians each time step to “age” it:

$$\underline{h}_a(n) = e^{j\Delta} \underline{h}(n), \quad \text{with } \Delta = \frac{2\pi}{L}$$

when our maximum sequence length (before reset) is  $L$

- **Idea:** “Warped RoPE” (WRoPE) for *arbitrarily long sequences*:

$$\Delta_n = \frac{2\pi n}{n + L}, \quad n = 0, 1, 2, \dots$$

(inspired by the *bilinear transform* used in digital filter design)

- A *blend* of *uniform* and *warped* rotations can be used:

$$\Delta_n = \begin{cases} \frac{\pi n}{L}, & n = 0, 1, 2, \dots, L - 1 \\ \pi + \frac{\pi n}{n+1}, & n = L, L + 1, L + 2, \dots \end{cases}$$

where  $L$  is now the *typical* sequence length (giving it more “space” in recall)



Background

Basic Idea

Architectures

Processing

• Perceptrons

• Sequences

• WRoPE

• Compressed Time

• Reservations

Attention

History Samples

## Searching a Time Range

- Reserving complex angle for positional encoding  $\Leftrightarrow$  *real* vocabulary embedding vectors
- Downstream weights and biases must be *complex*
- Complex angle discarded by absolute value when no longer needed
- Exact orthogonality is obtained for each of the  $L$  RoPE time-steps when there is no amplitude decay as in normal RNNs, *i.e.*,  $e^{jn\Delta} \underline{x}(n) \perp e^{jm\Delta} \underline{x}(n) \forall n \neq m$
- RoPE query vector for a vocabulary vector (“token”)  $\underline{w}$  any time between  $n_1$  to  $n_2$  is

$$\underline{q} = \sum_{n=n_1}^{n_2} e^{jn\Delta} \underline{w} = \underline{w} \frac{e^{j(n_2+1)\Delta} - e^{jn_1\Delta}}{e^{j\Delta} - 1} = \underline{w} e^{j\theta_{12}} \frac{\sin\left(\frac{n_2-n_1+1}{2}\Delta\right)}{\sin(\Delta/2)}$$



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

• Perceptrons

• Sequences

• WRoPE

• Compressed Time

• Reservations

[Attention](#)

[History Samples](#)

## Compressed Time Retrieval

- Recall that a vector  $\underline{w}$  is detected in the vector memory  $\underline{h}$  by the inner product  $\underline{w}^T \underline{x}$  exceeding some detection threshold  $b$ , i.e.,  $\underline{w}^T \underline{x} > b$
- The threshold  $b$  can be lowered to capture a wider range of similar vectors
- The captured vectors lie in a cone at angle  $\theta$  centered on the vector  $\underline{w}$ , and  $\cos(\theta) = b$
- In WRoPE, the angular rotation decreases as  $1/n$  or the like
- ...

This causes inner products to increase as vectors get squeezed together along the complex-angle arc.



Background

Basic Idea

Architectures

Processing

• Perceptrons

• Sequences

• WRoPE

• Compressed Time

• Reservations

Attention

History Samples

## Reserving Angular Dimensions

Reserving the radial dimension looks easy (e.g., RMSNorm every  $k$  gradient-descent steps)

How to reserve an *arc dimension* for [W]RoPE and perhaps other purposes?

- RoPE: Effectively reserves *complex angle*:  $\underline{h}_a(n) = e^{j\Delta} \underline{h}(n)$   
(data size doubled:  $\underline{x} \rightarrow (\underline{x}, \underline{0})$ )
- A zero imaginary part in the vocabulary embedding reserves complex angle
- *Quaternions* give two more angles  $\Leftrightarrow$  embedding size quadrupled:  $\underline{x} \rightarrow (\underline{x}, \underline{0}, \underline{0}, \underline{0})$ 
  - $\underline{h}_a(n) = \mathbf{q} \underline{h}(n)$ , where, from Wikipedia:  
$$\mathbf{q} = e^{\frac{\Delta}{2}(\underline{a}_x \mathbf{i} + \underline{a}_y \mathbf{j} + \underline{a}_z \mathbf{k})} = \cos \frac{\Delta}{2} + (\underline{a}_x \mathbf{i} + \underline{a}_y \mathbf{j} + \underline{a}_z \mathbf{k}) \sin \frac{\Delta}{2} = \cos \frac{\Delta}{2} + \underline{a} \sin \frac{\Delta}{2}$$
- See the Cayley-Dickson Construction for  $2^k - 1$  angles,  $k = 1, 2, 3, \dots$
- Trivial in *polar coordinates* (“write-protect” reserved dimensions during gradient step), but then back-propagation must be rewritten for polar coordinates
- Find a suitable method of *constrained gradient descent*
- Train (or calculate) a *pointwise MLP* that “de-rotates”  $\underline{x} + \mu \nabla$  given also  $\underline{x}$
- Use *reserved translational dimension(s)* instead (“TraPE”) (omitted from RMSNorm)



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

# Attention



Background

Basic Idea

Architectures

Processing

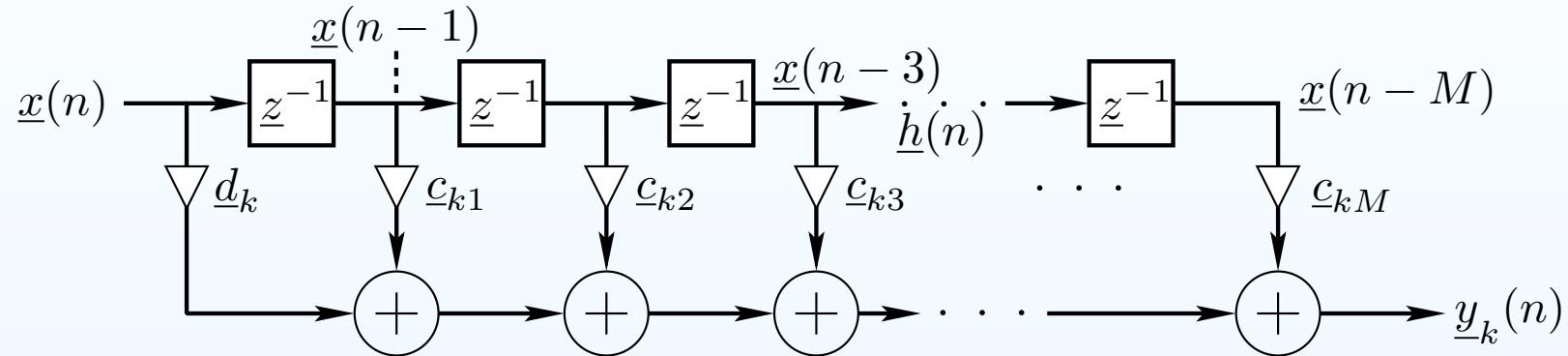
Attention

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- TransMamba
- Direct Forms
- Plan
- Hypersphere

History Samples

## Attention Layer

Idea: Also use *FIR Filtering* (SSM State Expansion Factor  $M$ ,  $\mathbf{A}$  subdiagonal):



Separately learnable FIR coefficient matrices  $d_k[\underline{x}(n)], c_j[\underline{x}(n-j), k]$ , depending on:

1. input position  $j$  in the input sequence (“context buffer” or “expanded state” + [W]RoPE)
2. input vector  $\underline{x}(n-j)$ ,  $j = 0, 1, 2, \dots, M$
3. output-position  $k$  being computed,  $k = 0, 1, 2, \dots, M$  ( $M + 1$  outputs)

Idea: Add *relevance gating* suppressing unimportant inputs to each output (“attention”)

Idea: Create new embedding vectors as sums of relevant input vectors (“attention”)

Idea: Measure relevance using an *inner product* between the output and input positions (“dot-product attention”)



Background

Basic Idea

Architectures

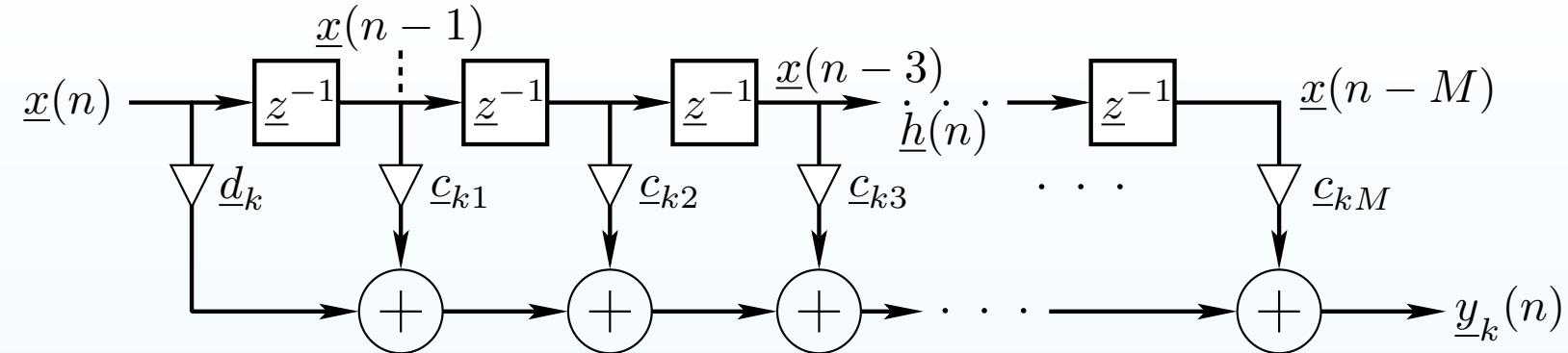
Processing

Attention

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- TransMamba
- Direct Forms
- Plan
- Hypersphere

History Samples

## Dot-Product Attention



### Relevance Gating

Let  $\underline{x}_k$  denote  $\underline{x}(n - k)$

The contribution from input  $\underline{x}_j$  to the nonlinear FIR sum for output  $\underline{y}_k$  can be calculated as

$$\underline{c}_{kj}\underline{x}_j = \left[ \left( \sum_{m \in \mathcal{R}(\underline{x}_k)} \underline{x}_m \right)^T \underline{x}_j \right] \underline{x}_j$$

or more generally  $\underline{c}_{kj} = Q_k^T \underline{x}_j$ , where

$Q_k(\underline{x}_k, k)$  is called the *query vector* for position  $k$  in the input sequence

The query  $Q_k$  can be a sum of *all vectors supported in the attention sum*:

$$Q_k = \underline{x}_k + \underline{x}_{m_1} + \dots + \underline{x}_{m_k}$$

$\Rightarrow (Q_k^T \underline{x}_j) \underline{x}_j \approx \underline{x}_j$ , if  $\underline{x}_j$  is similar to *any vector* in the query sum.



[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

- Attention
- Dot-Product Attention
- Multi-Head Attention

● Unified State Space

● TransMamba

● Direct Forms

● Plan

● Hypersphere

[History Samples](#)

## Multi-Head Attention

**Idea:** To support multiple meaning possibilities, *partition the model space* into parallel independent *attention calculations* (“multi-head attention”)

- Each *attention head* can form an independent input interpretation
- Useful for *ambiguous* sequences, especially in the lower layers
- Also introduced in the Transformer paper (2017)

Now we need *down-projections* of the relevance-calculation components

⇒ relevance of input  $j$  to output  $k$  in attention-head  $l$  becomes proportional to

$$\underline{c}_{kj} \underline{x}_j = (Q_k^T \underline{x}_j) \underline{x}_j \longrightarrow \underline{c}_{lkj} \underline{x}_{lj} = [Q_{lk}^T(\underline{x}_k) K_{lj}(\underline{x}_j)] V_{lj}(\underline{x}_j)$$

where  $Q_{lk}$  (“query”),  $K_{lj}$  (“key”), and  $V_{lj}$  (“value”) vectors are learned *down-projections* of the input  $\underline{x}_j$  for each attention-head  $l$  and for all sequence indices  $j$  and  $k$  in the context buffer (“Transformer”)

Other useful generalizations can be imagined for these learned (Q,K,V) vectors, such as grouping grammatical functions, creating new model-space regions, etc.





[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

- Attention
- Dot-Product Attention
- Multi-Head Attention
- **Unified State Space**
- TransMamba
- Direct Forms
- Plan
- Hypersphere

[History Samples](#)

## State Space Unification of Transformers and GRNNs

$$\mathbf{A}_T = \underline{e}^{j\Delta_n} \underbrace{\begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ \underline{1} & 0 & \cdots & 0 & 0 \\ 0 & \underline{1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \underline{1} & 0 \end{pmatrix}}_{\text{Transformer}}$$

$$\mathbf{A}_M = \underline{a}_n \underline{e}^{j\Delta_n} \underbrace{\begin{pmatrix} \underline{1} & 0 & \cdots & 0 \\ 0 & \underline{1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \underline{1} \end{pmatrix}}_{\text{Mamba-2 style RNN + [W]RoPE}}$$

$$\mathbf{A}_{TM} = \underline{e}^{j\Delta_n} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \underline{1} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \underline{1} & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \underline{1} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & \underline{\beta}_1(n) & \underline{a}_n & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \underline{\beta}_{N_M}(n) & 0 & \cdots & \underline{a}_n \end{pmatrix}$$

$$\mathbf{B}_{TM} = \begin{pmatrix} \underline{b}_1(n) \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$



[Background](#)

[Basic Idea](#)

[Architectures](#)

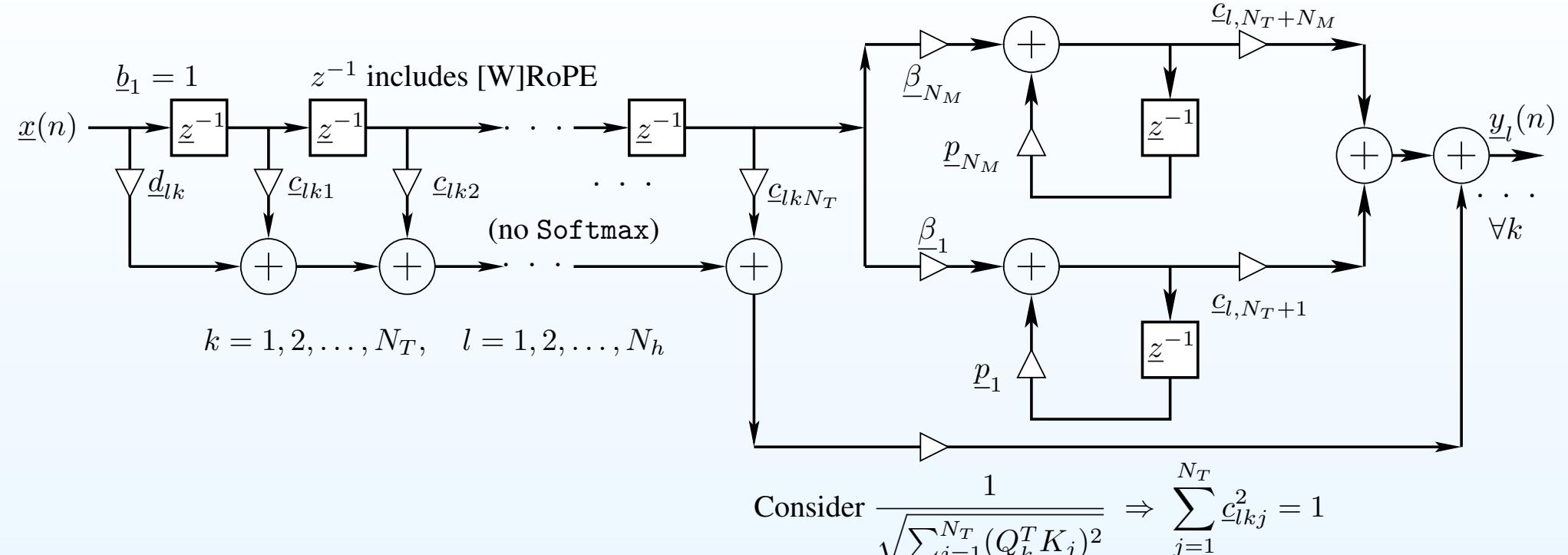
[Processing](#)

[Attention](#)

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- **TransMamba**
- Direct Forms
- Plan
- Hypersphere

[History Samples](#)

## Transformer followed by GRNN with 2x State Expansion (like Mamba)



TransMamba





[Background](#)

[Basic Idea](#)

[Architectures](#)

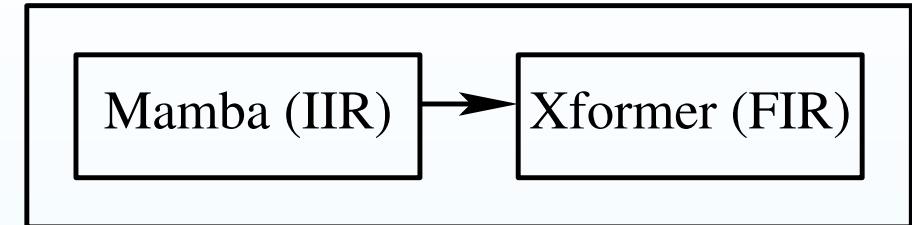
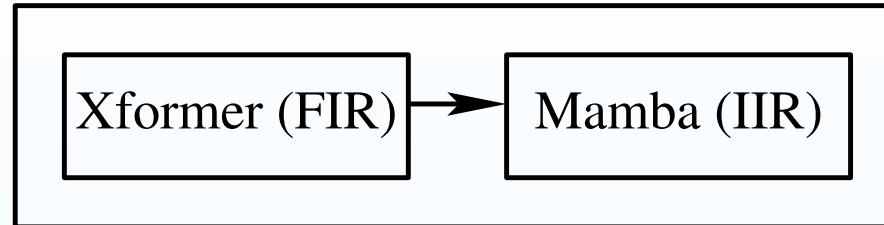
[Processing](#)

[Attention](#)

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- TransMamba
- Direct Forms
- Plan
- Hypersphere

[History Samples](#)

## “Direct Form I” or “Direct Form II”?



### XMamba (“DF-I”) versus MambaX (“DF-II”)

- Perfect short-term memory
- Fuzzy, fading, long-term memory
- Like Infini-Attention

- Fading short-term memory
- Limited long-term memory
- Also like the brain

Maybe use both?



Background

Basic Idea

Architectures

Processing

Attention

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- TransMamba
- Direct Forms
- Plan
- Hypersphere

History Samples

## Next Steps

- Try to improve [Trans]Mamba[-2] on small synthetic datasets testing *memory*
  - Vocabulary embeddings trained to the unit hypersphere (e.g., word2sphere)
  - Memory *duration* and *reset* functions separately trained and implemented
  - Initial *biases* at  $\underline{0}$  versus  $L/N$  etc.
  - Do *power normalization* in place of RMSNorm where possible (efficiency)
  - Try *power normalized attention* in place of  $1/\sqrt{d_h}$  and Softmax (efficiency)
  - Adapt *model dimension* to *layer width* at each level (efficiency)
  - Warped Rotational Positional Encoding (WRoPE)
  - Translational Positional Encoding (TraPE) in its own head (no RMSNorm)
  - Explore other “Control Heads” that flow along purely for “conditioning” like TraPE
- Progress to date:
  - New synthetic benchmarks analogous to “needle in a haystack”
  - Adapted Andrej Karpathy’s makemore code, adding Mamba and new benchmarks
  - Four papers started, aiming for Arxiv, GitHub, “AI social media,” blog
- Feel free to take over any of these! (and LMK so I can do something else)





[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

- Attention
- Dot-Product Attention
- Multi-Head Attention
- Unified State Space
- TransMamba
- Direct Forms
- Plan
- Hypersphere

[History Samples](#)

## Thanks for your Attention!





[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

## Sequence Modeling Snapshots



## LSTM and GRU

Background

Basic Idea

Architectures

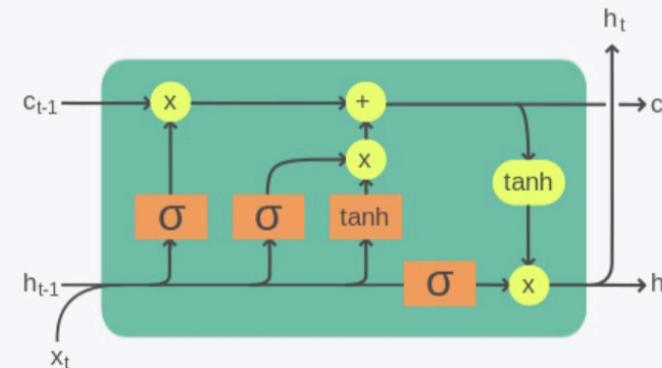
Processing

Attention

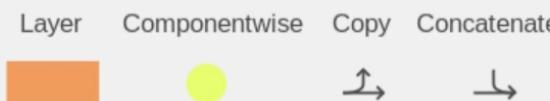
History Samples

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

1997: LSTM

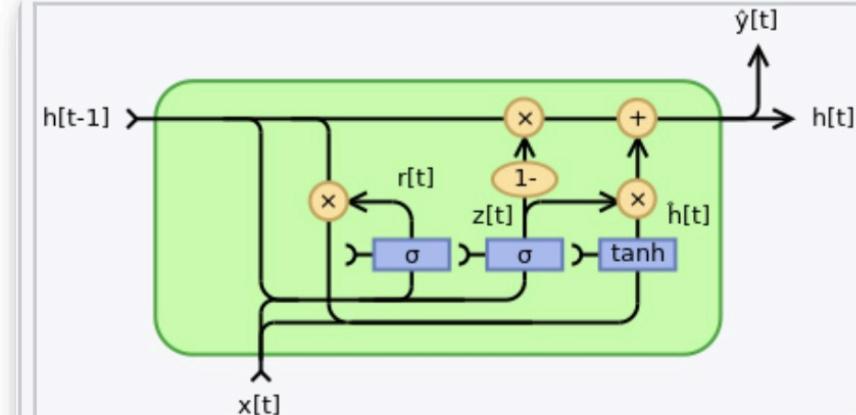


Legend:



The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time.

2014: GRU



Gated Recurrent Unit, fully gated version



Background

Basic Idea

Architectures

Processing

Attention

History Samples

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

## Structured State Space and Mamba

2023: Mamba (S6)

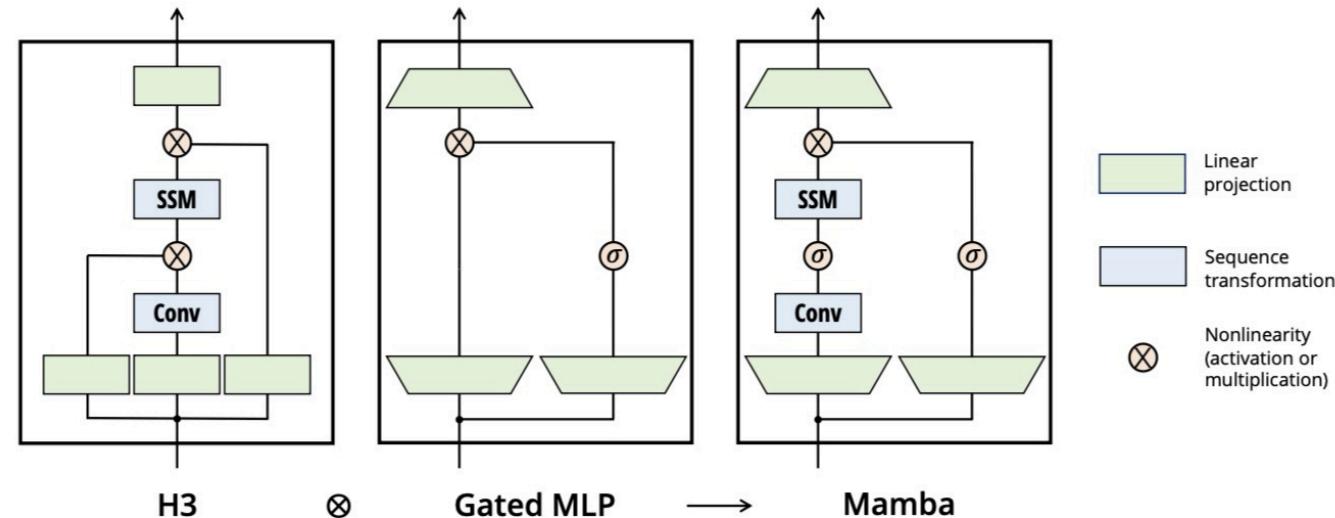


Figure 2: (Architecture.) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogeneously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For  $\sigma$  we use the SiLU / Swish activation (Hendrycks & Gimpel, 2016; Ramachandran et al., 2017).



# Hawk and Griffin

[Background](#)

[Basic Idea](#)

[Architectures](#)

[Processing](#)

[Attention](#)

[History Samples](#)

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models

## 2024: Hawk & Griffin

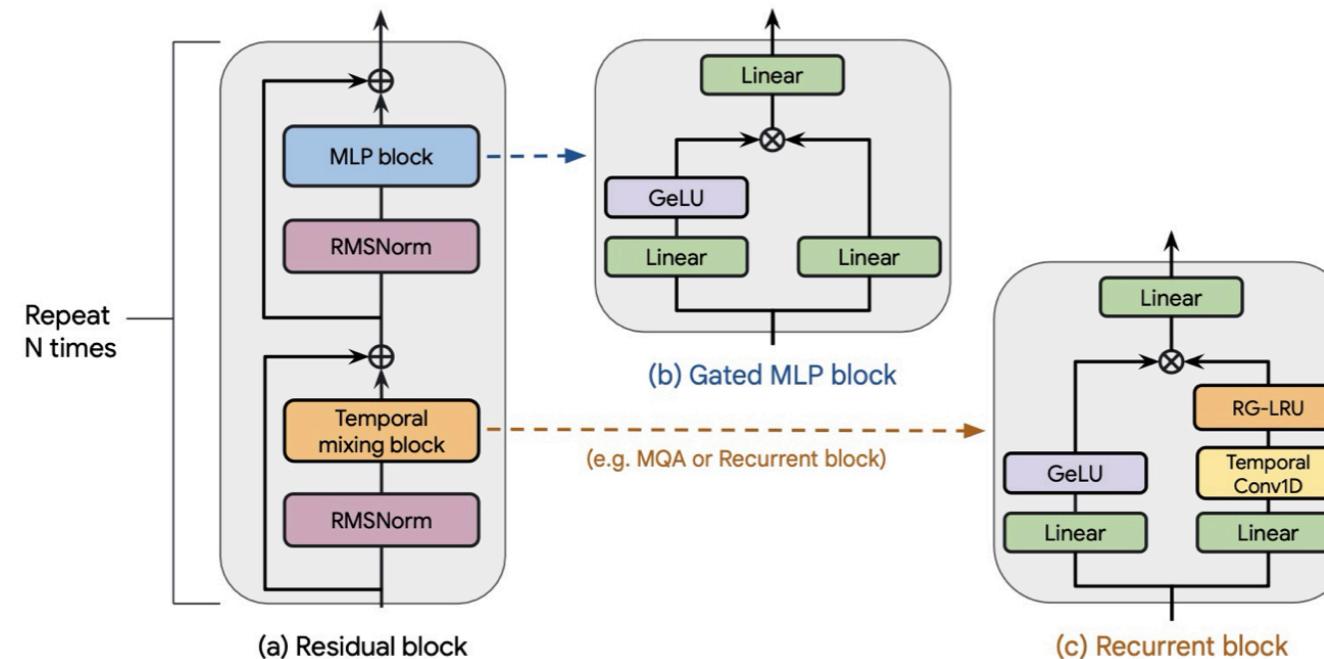


Figure 2 | a) The main backbone of our mode architecture is the residual block, which is stacked  $N$  times. b) The gated MLP block that we use. c) The recurrent block that we propose as an alternative to Multi Query Attention (MQA). It uses our proposed RG-LRU layer, defined in Section 2.4.



Background

Basic Idea

Architectures

Processing

Attention

History Samples

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

## Gated “Linear” RNNs with State Expansion

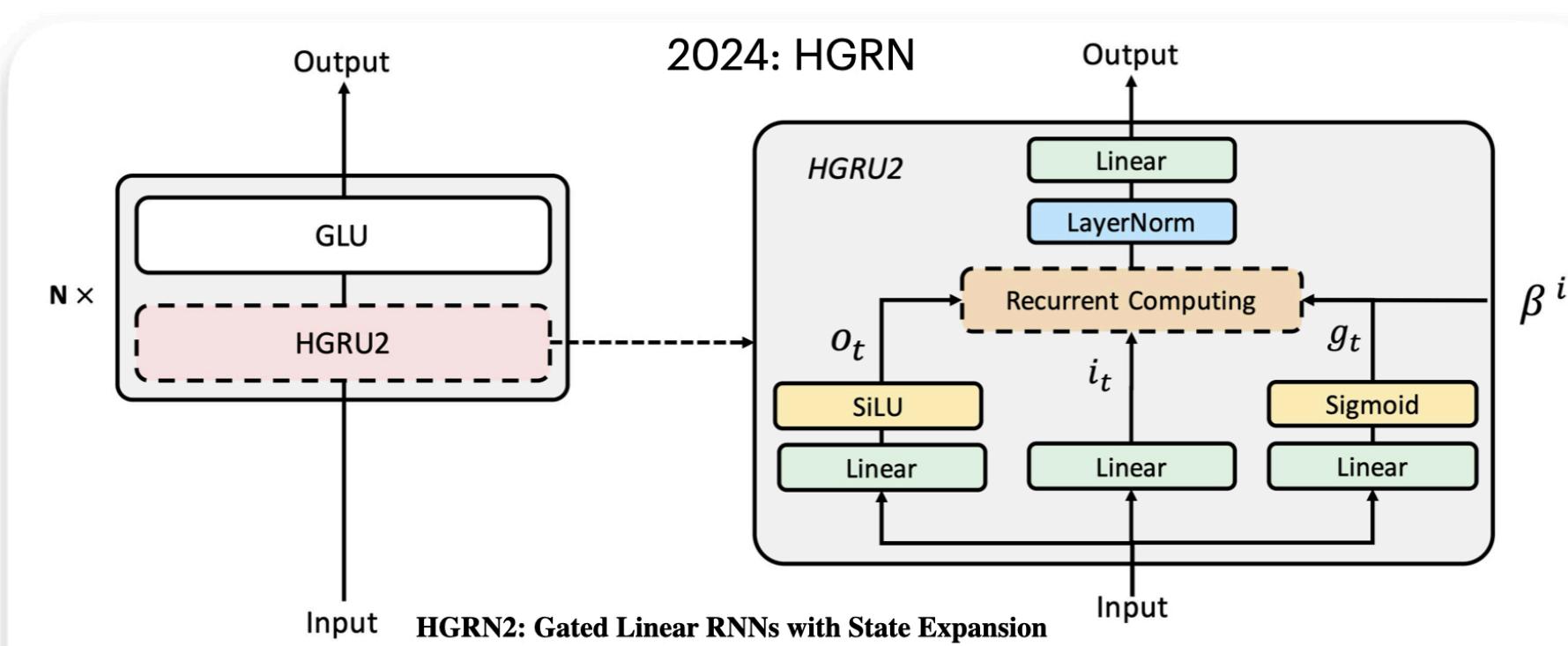


Figure 1: **The neural architecture of HGRN2.** Each HGRN2 layer includes a token mixer layer HGRU2 and a channel mixerlayer GLU. HGRU2 employs recurrent computation through Eq. 3, where  $i_t$  is the input vector,  $g_t$  is the forget gate (not lower bounded),  $\beta^i$  is the lower bound of the forget gate value,  $o_t$  is the output gate for layer  $i$ .



# RWKV, Eagle, Finch

Background

Basic Idea

Architectures

Processing

Attention

History Samples

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

## 2024: Eagle-Finch RWKV

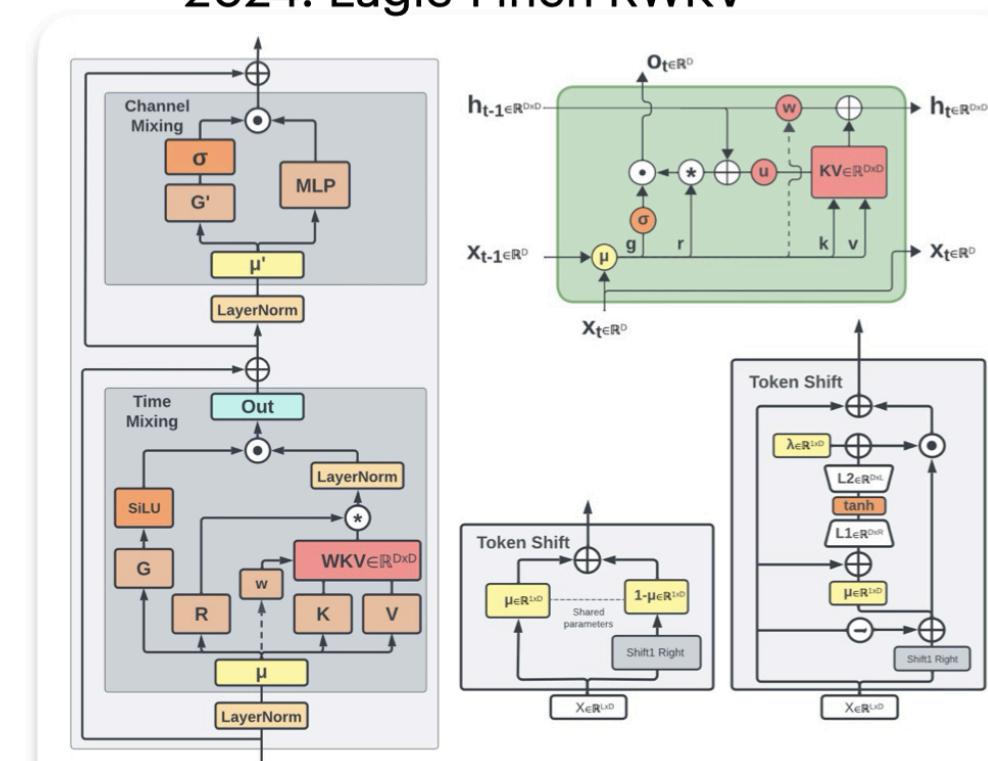


Figure 1: RWKV architecture overview. **Left:** time-mixing and channel-mixing blocks; **top-right:** RWKV time-mixing block as RNN cell; **center-bottom:** token-shift module in FeedForward module and Eagle time-mixing; **bottom-right:** token-shift module in Finch time-mixing. All shape annotations assume a single head for simplicity. Dashed arrows (left, top-right) indicate a connection in Finch, but not in Eagle.



Background

Basic Idea

Architectures

Processing

Attention

History Samples

- LSTM & GRU
- SSM & Mamba
- Hawk & Griffin
- HGRN2
- RWKV+
- Hybrid

## Jamba, Zamba, & Samba Hybrid Architectures (Mamba then Attention)

Jamba (3/2024)      Zamba (5/2024)      Samba (6/2024)

