

Alternative Models

Structured Discrete SSMs: S4 & Mamba

ELL8299 · ELL881 · AIL861



Sourish Dasgupta

Associate Professor, DAU, Gandhinagar
<https://daiict.ac.in/faculty/sourish-dasgupta>

State Space Machines – *Language as a Diffusive Field*

Core idea. Instead of updating memory in discrete jumps (as in RWKV or LSTM), a State Space Model (SSM) treats hidden meaning as a *continuously evolving field*:

$$\frac{dx}{dt} = Ax(t) + Bu(t).$$

- $x(t)$ — the latent semantic field (what the model “feels” at any instant)
- $Ax(t)$ — how that field drifts or decays on its own (internal physics)
- $Bu(t)$ — how the current token *nudges* or perturbs the field



Recap: Continuous SSM

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t)$$

Meaning of terms

- $x(t)$ – internal memory state (semantic context)
- $u(t)$ – input (embedding of current token)
- A – memory dynamics (decay/oscillation)
- B, C – input and output mappings

Linguistic analogy: Each token perturbs an ongoing semantic “stream.” A decides how the memory of previous tokens (e.g., “John loves”) continues to influence new ones.



Every Eigenvalue is a *Semantic Rhythm*

Let $\lambda_i = a_i + ib_i$ be eigenvalues of A .

$$e^{\lambda_i t} = e^{a_i t} (\cos(b_i t) + i \sin(b_i t))$$

- a_i (real part): rate of memory decay.
- b_i (imag part): oscillation—how memory recurs.

Linguistic intuition:

- $a_i < 0$: “Mary” fades slowly—topic decay.
- $b_i \neq 0$: “subject–verb–object” cycles reappear rhythmically.



Discretization of Continuous SSM

Tokens arrive at discrete times: $t = 0, \Delta, 2\Delta, \dots$

$$x_{t+1} = \bar{A}x_t + \bar{B}u_t, \quad \bar{A} = e^{A\Delta}, \quad \bar{B} = \int_0^\Delta e^{A\tau} B d\tau$$

Linguistic view: Each word samples the continuous semantic trajectory at one instant—quantizing the continuous evolution of meaning.



Convolutional Memory (instead of Sequential)

$$y_t = C \sum_{k=1}^t \bar{A}^{k-1} \bar{B} u_{t-k} \Rightarrow y = K * u, \quad K_k = C \bar{A}^{k-1} \bar{B}.$$

Meaning builds from overlapping ripples.



Making computation fast via FFT

Problem: Direct convolution is $O(T^2)$.

FFT trick:

$$\text{FFT}(y) = \text{FFT}(K) \odot \text{FFT}(u)$$

$$y = \text{IFFT}(\text{FFT}(K) \odot \text{FFT}(u))$$

- Compute convolution in frequency domain in $O(T \log T)$.
- Each frequency = a rhythm of semantic evolution.

Language view:

Instead of replaying every echo word-by-word, FFT lets the model “play all rhythms of meaning at once.”



Is FFT-based Kernelized Discrete SSMs enough?

Question: We discretized the continuous SSM to get

$$x_{t+1} = \bar{A}x_t + \bar{B}u_t, \quad y_t = Cx_t + Du_t.$$

Does this fix the instability and inefficiency?

Answer: Not really.

- Discretization changes *how* we step through time, but not *how stable* those steps are.
- Problems of learning, numerical drift, and efficiency remain.



Instability in learning A , B , C Matrices

Even after discretization:

$$x_{t+1} = \bar{A}x_t + \bar{B}u_t$$

Stability condition

Result: exploding or vanishing hidden states even in discrete form.

Linguistic view: Some words may dominate memory endlessly (“John John John...”), while others vanish too quickly (“City” forgotten instantly).

eigenvalues outside the unit circle.



Exponential Growth/Decay with k

The kernel still involves repeated powers of \bar{A} :

$$K_k = C \bar{A}^{k-1} \bar{B}.$$

- If $|\lambda_i| < 1$, memory decays exponentially—model forgets too fast.
- If $|\lambda_i| > 1$, small errors explode—model becomes unstable.
- Discretization does not remove this exponential sensitivity; it merely reparameterizes it.

Linguistic analogy: A stable model should let “Mary” fade gradually. But with poor eigenvalues, her influence either disappears too soon or echoes unnaturally forever.



Computational cost depends on diagonalizability of A

To compute the convolution:

$$y_t = \sum_{k=1}^t C \bar{A}^{k-1} \bar{B} u_{t-k}.$$

- Still requires $O(Tn^2)$ multiplications if \bar{A} is unstructured.
- Large sequences ($T > 10^3$) become prohibitively slow.
- FFT reduces cost to $O(T \log T)$, but efficiency also depends on how easily \bar{A} can be diagonalized.



So far, we were dealing with **Eigen Decomposition**

Type	Mathematical form	When valid / what it implies
Eigen-decomposition	$A = V\Lambda V^{-1}$	Any diagonalizable A ; <u>V need not be orthogonal.</u>
Spectral decomposition	$A = V\Lambda V^*$	Only for <u>normal</u> matrices ($AA^* = A^*A$); V unitary \Rightarrow stable, orthogonal modes.

Table: Spectral vs. Eigen decomposition. Spectral adds orthogonality and stability.



Takeaway: Structure is still needed

Discretization \neq Regularization

It only allows token-level updates; it does not guarantee stability, memory balance, or computational efficiency.

Therefore:

- We need a structured A with eigenvalues guaranteed to stay inside the stability region.
- We need efficient spectral computation of K_k .
- These goals motivate the **Structured State Space Sequence Model (S4)**.



Desiderata ...

- **Remember** the past a long time without exploding/forgetting.
- **Update** the memory online as each token arrives.
- **Be efficient** on long sequences (sub-quadratic preferred).
- **Be interpretable** enough to reason about “what” is remembered.

Linguistic interpretation

A reader keeps: (i) *topic* (slow memory), (ii) *phrase rhythm* (medium), (iii) *connectors/markers* (fast). We want the model to do the same.



How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections

Albert Gu^{*†}, Isys Johnson^{*‡}, Aman Timalsina[‡], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University

[†]albertgu@stanford.edu, chrismre@cs.stanford.edu

[‡]Department of Computer Science and Engineering, University at Buffalo

[‡]{isysjohn, amantima, atri}@buffalo.edu



Structured Parameterization of A – The HiPPO way

Continuously summarize the entire past $u(\tau)$ into a few running summaries $c_k(t)$:

$$c_k(t) \approx \int_0^t u(\tau) p_k(\tau) d\tau \quad \text{where each } p_k(\tau) \text{ is a smooth shape.}$$

- Each p_k is a simple, smooth “shape” (like a wave, slope, or curve) that looks for a specific trend in the past signal.
- These shapes are chosen to be orthogonal — meaning they don’t overlap or interfere with one another.
- c_0 = average topic; c_1 = recent trend; c_2, c_3, \dots = finer, short-term details.



HiPPO (High-Order Polynomial Projection Operator) - The intuition

Continuously summarize the entire past $u(\tau)$ into a few running summaries $c_k(t)$:

$$c_k(t) \approx \int_0^t u(\tau) p_k(\tau) d\tau \quad \text{where each } p_k(\tau) \text{ is a smooth shape.}$$

Linguistic interpretation

Think of HiPPO as keeping a few “dials” tuned to different aspects of meaning: a slow dial for topic, a medium one for phrase rhythm, and a fast one for connectors.



Why orthogonal polynomials?

- **Orthogonality** = memory channels do not interfere.
- The induced dynamics matrix A_{HiPPO} has $\text{Re } \lambda_i < 0 \Rightarrow$ **stability**.
- Each dimension acts like a **different memory speed**.

Goal of HiPPO

Design $A_{\text{HiPPO}}, B_{\text{HiPPO}}$ so $c(t)$ tracks these coefficients *online*, without re-integrating from scratch.



The HiPPO ODE

We wish to maintain

$$c_k(t) \approx \int_0^t u(\tau) p_k(\tau) d\tau.$$

$$\dot{c}(t) = A_{\text{HiPPO}} c(t) + B_{\text{HiPPO}} u(t)$$

where

$$(A_{\text{HiPPO}})_{kn} = - \begin{cases} \sqrt{(2k+1)(2n+1)}, & n < k, \\ k+1, & n = k, \\ 0, & n > k, \end{cases} \quad B_{\text{HiPPO},k} = \sqrt{2k+1}.$$



Getting to the HiPPO ODE - *From Legendre to Orthonormal Basis*

The standard Legendre polynomials $P_k(x)$ are orthogonal on $[-1, 1]$:

$$\int_{-1}^1 P_k(x) P_n(x) dx = \frac{2}{2k+1} \delta_{kn}.$$

To adapt them to $[0, 1]$, change variables:

$$x = 2s - 1 \quad \Rightarrow \quad dx = 2 ds.$$

Then,

$$\int_0^1 P_k(2s-1) P_n(2s-1) ds = \frac{1}{2} \int_{-1}^1 P_k(x) P_n(x) dx = \frac{1}{2k+1} \delta_{kn}.$$

Intuition

Each $\bar{p}_k(s)$ is a “shape” that captures a specific time scale: slow ($k = 0$) for topics, medium for phrases, fast for connectors.



Getting to the HiPPO ODE - *Normalizing Time-Interval*

We begin with the continuous projection goal:

$$c_k(t) = \int_0^t u(\tau) p_k(\tau, t) d\tau.$$

Here p_k are basis polynomials defined over a growing domain $[0, t]$.

Problem: As t increases, the integration limits change. We need a fixed interval to maintain an *online* memory update.

Solution: Normalize the time variable:

Define time-independent basis functions on $[0, 1]$:

$$\bar{p}_k(s) := p_k(ts, t),$$

so that we can work with fixed, orthonormal functions as t evolves.



Getting to the HiPPO ODE - *Normalized Projection*

We start from the continuous projection goal:

$$c_k(t) = \int_0^t u(\tau) p_k(\tau, t) d\tau,$$

where p_k are orthogonal polynomials (Legendre).

Normalize the interval:

$$s = \frac{\tau}{t},$$

$$\bar{p}_k(s) = \sqrt{2k+1} P_k(2s-1).$$

Then

$$c_k(t) = t \int_0^1 u(ts) \bar{p}_k(s) ds.$$

Goal: Differentiate $c_k(t)$ to get an ODE of the form

$$\dot{c}(t) = A_{\text{HiPPO}} c(t) + B_{\text{HiPPO}} u(t).$$



Getting to the HiPPO ODE - *Differential update*

Linguistic interpretation

Each token injects new signal $u(t)\bar{p}_k(1)$, while the derivative term shifts and fades previous memories according to how fast each polynomial shape evolves.

To remove $u'(ts)$, integrate by parts:

$$t \int_0^1 s u'(ts) \bar{p}_k(s) ds = \underline{u(t)\bar{p}_k(1)} - \int_0^1 u(ts) \frac{d}{ds} [s \bar{p}_k(s)] ds.$$

Hence:

$$\dot{c}_k(t) = \underline{u(t)\bar{p}_k(1)} + \int_0^1 u(ts) \left(\bar{p}_k(s) - \frac{d}{ds} [s \bar{p}_k(s)] \right) ds.$$



Getting to the HiPPO ODE - *The final form*

Legendre polynomials satisfy:

$$\frac{d}{ds}[s \bar{p}_k(s)] = (k+1)\bar{p}_k(s) + \sum_{n=0}^{k-1} \sqrt{(2k+1)(2n+1)} \bar{p}_n(s).$$

Substitute into the integral:

$$\dot{c}_k(t) = \sqrt{2k+1} u(t) - \left[(k+1)c_k(t) + \sum_{n=0}^{k-1} \sqrt{(2k+1)(2n+1)} c_n(t) \right].$$

Collecting all c_k :

$$\dot{c}(t) = A_{\text{HiPPO}} c(t) + B_{\text{HiPPO}} u(t)$$

where

$$(A_{\text{HiPPO}})_{kn} = - \begin{cases} \sqrt{(2k+1)(2n+1)}, & n < k, \\ k+1, & n = k, \\ 0, & n > k, \end{cases} \quad B_{\text{HiPPO},k} = \sqrt{2k+1}.$$



Interpreting the HiPPO matrices

$$(A_{\text{HiPPO}})_{kn} = - \begin{cases} \sqrt{(2k+1)(2n+1)}, & n < k, \\ k+1, & n = k, \\ 0, & n > k, \end{cases} \quad B_{\text{HiPPO},k} = \sqrt{2k+1}.$$

Interpretation

- Upper-triangular A_{HiPPO} : slow modes feed fast ones (context \rightarrow connectors).
- Negative diagonal: ensures stable decay.
- B_{HiPPO} : injects new token into all memory scales.



Still continuous!

$$\dot{x}(t) = A_H x(t) + B_H u(t), \quad y(t) = C_H x(t),$$

with $x(t) \equiv c(t)$ and $A_H \equiv A_{\text{HiPPO}}$.

- Stable by construction: $\text{Re } \lambda_i(A_H) < 0$.
- Each state dimension = one **memory mode** (speed).

Linguistic interpretation

Different channels store: topic, phrase rhythm, connectors — simultaneously.



Published as a conference paper at ICLR 2022

EFFICIENTLY MODELING LONG SEQUENCES WITH STRUCTURED STATE SPACES

Albert Gu & Karan Goel & Christopher Ré

Department of Computer Science, Stanford University

{albertgu,krng}@stanford.edu, chrismre@cs.stanford.edu



Adopt the good things of HiPPO but make it *normal & light*

Direct eigen-decomposition can be ill-conditioned. S4 uses **Normal + Low-Rank**:

$$A_H = N + p q^T, \quad N \text{ normal} \Rightarrow N = V \Lambda V^*.$$

Change basis $z = V^* x$ and define

$$B_0 = V^* B_H, \quad C_0 = C_H V, \quad \tilde{p} = V^* p, \quad \tilde{q} = V^* q.$$

Dynamics become

$$\dot{z} = (\Lambda + \tilde{p} \tilde{q}^*) z + B_0 u, \quad y = C_0 z,$$

i.e., **diagonal + low-rank** in a numerically nice basis.



Discretize the SSM

For step Δ :

$$\bar{A} = e^{A\Delta}, \quad \bar{B} = \int_0^\Delta e^{A\tau} B d\tau.$$

With $A = \Lambda + \tilde{p}\tilde{q}^*$ (diagonal + low-rank), exponentials/inverses are handled with Cauchy/ Woodbury identities efficiently.

Plain words

Turn the smooth memory flow into per-token updates — still stable, still multi-speed.



Applying FFT

Time domain:

$$y_t = \sum_{k \geq 1} K_k u_{t-k}, \quad K_k = C \bar{A}^{k-1} \bar{B}.$$

Frequency domain (used for fast convolution):

$$\hat{K}(\omega) = C (e^{i\omega} I - \bar{A})^{-1} \bar{B}.$$

Diagonal + low-rank \Rightarrow fast inverse via Woodbury:

$$(D - UV^T)^{-1} = D^{-1} + D^{-1}U(I - V^T D^{-1}U)^{-1}V^T D^{-1},$$

with D diagonal across modes.



The final contextual embedding

$$y = \text{IFFT} \left(\hat{K} \odot \text{FFT}(u) \right).$$

Complexity

$O(nr + r^3)$ per frequency for the inverse (typically $r \ll n$), overall $O(T \log T)$.



The S4 Architecture Pipeline

- 1 Project tokens $\rightarrow u$.
- 2 Spectral filtering: $y_{\text{SSM}} = K * u$ via FFT (kernel from $A = \Lambda + \tilde{p}\tilde{q}^*$).
- 3 Optional gate/normalization.
- 4 MLP + residual.



Performance comparison

Table 2: Benchmarks vs. efficient Transformers

	LENGTH 1024		LENGTH 4096	
	Speed	Mem.	Speed	Mem.
Transformer	1×	1×	1×	1×
Performer	1.23×	<u>0.43</u> ×	3.79×	<u>0.086</u> ×
Linear Trans.	1.58 ×	0.37 ×	5.35 ×	0.067 ×
S4	1.58 ×	<u>0.43</u> ×	<u>5.19</u> ×	0.091×

Observation

S4 achieves comparable or better throughput than linear-time Transformers while retaining strong long-sequence stability and memory efficiency.



Performance comparison

1. Scaling behaviour.

- Transformer complexity: $\mathcal{O}(T^2)$ in both time and memory.
- S4 complexity: $\mathcal{O}(T \log T)$ (via FFT-based convolution).

2. Why S4 matches Linear Transformers in speed.

- Low-rank + diagonal structure of $A = \Lambda + pq^T$ enables fast kernel computation.
- FFT parallelizes convolution across tokens.

3. Memory advantage.

- Constant-size state ($x_t \in \mathbb{R}^n$) $\Rightarrow O(1)$ memory per token.
- No need to store pairwise attention scores.



Performance comparison

Table 3: (Long Range Arena) Accuracy on full suite of LRA tasks. (Top) Original Transformer variants in LRA. Full results in Appendix D.2. (Bottom) Other models reported in the literature.

MODEL	LISTOPS	TEXT	RETRIEVAL	IMAGE	PATHFINDER	PATH-X	AVG
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	✗	50.46
Performer	18.01	<u>65.40</u>	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	<u>77.80</u>	✗	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	<u>79.29</u>	<u>47.38</u>	77.72	✗	<u>59.37</u>
S4	58.35	76.02	87.09	87.26	86.05	88.10	80.48

Takeaway

S4 combines **spectral efficiency** with **robust generalization**. It keeps long-term dependencies without attention and adapts across modalities — text, audio, and symbolic reasoning.



Where state-of-the-art models fall short

Prompt: How should a model remember and reason through —
“John promised Mary he would not visit Paris yesterday”?

- Each architecture answers this question differently.
- Some remember too much; others forget too fast.



Where state-of-the-art models fall short - *Transformers*

Claim: "Every word looks at every other."

Mechanism:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V$$

- **MLA / GQA (DeepSeek V3, Llama 4):** compress or share keys/values to save memory.
- **Sliding Window Attention (Gemma 3):** restrict attention to nearby context.

Linguistic lens:

- "He" can talk directly to "John"—great for dependencies.
- But every word joins the conversation—costly and noisy.



Where state-of-the-art models fall short - *RNN-styled*

Claim: "We read like humans — one word at a time."

$$h_t = f(W_h h_{t-1} + W_x x_t)$$

- Keeps a running summary (hidden state) through time.
- Suffers from vanishing/exploding gradients — forgets long dependencies.

Linguistic view:

- When "not" appears, the memory of "promised" has already faded.
- Like a storyteller who loses the subject halfway through.



Where state-of-the-art models fall short - *Linear Attention*

Claim: “We keep attention’s awareness but drop its cost.”

$$\text{Attn}(Q, K, V) \approx \phi(Q) \left(\phi(K)^T V \right)$$

Qwen3-Next, Kimi Linear, MiniMax M1

- Replace softmax with kernel or DeltaNet approximations.

Linguistic lens:

- The *speed-reader* — skims the text quickly but only partially catches the nuance of “not” or “Paris.”



Where state-of-the-art models fall short - *MoE*

Claim: “Instead of one big brain, we use many small experts.”

$$y_t = \sum_{i \in \text{active}} g_{t,i} f_i(u_t)$$

DeepSeek V3/R1, Llama 4, GLM 4.5, Kimi K2

• Each token activates a handful of specialized feed forward experts

Linguistic lens:

- A newsroom of specialists: “promised” wakes the verb expert, “not” the negation expert.
- But experts don’t listen to each other across time.



Where state-of-the-art models fall short - S4

Claim: “We model memory as continuous dynamics.”

$$\dot{x}(t) = Ax(t) + Bu(t)$$

- Long-range, stable, mathematically elegant.
- But A, B, C are fixed—no awareness of what each token means.

Linguistic lens:

- Like an orchestra that keeps playing one rhythm even when “not” should invert the tune.



What all the SOTA models miss ...

Common Gaps

Linguistic Analogy

- Transformer: Noisy

Core Limitation

Memory everywhere is still **structure-driven**, not **meaning-driven**. No mechanism decides dynamically what to keep or forget based on token semantics.

- MoE: powerful but disconnected.
- MoE: Panel of experts without memory.



Published as a conference paper at COLM 2024

Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu*
Machine Learning Department
Carnegie Mellon University
agu@cs.cmu.edu

Tri Dao*
Department of Computer Science
Princeton University
tri@tridao.me



S4 summarized

- Continuous SSM: $\dot{x}(t) = Ax(t) + Bu(t)$, $y(t) = Cx(t) + Du(t)$
- Discretize at token rate Δ : $x_{t+1} = \bar{A}x_t + \bar{B}u_t$, $y_t = Cx_t + Du_t$
- Convolutional view: $y = K * u$, with $K_k = C\bar{A}^{k-1}\bar{B}$
- Spectral efficiency: $\hat{K}(\omega) = C(e^{i\omega}I - \bar{A})^{-1}\bar{B}$
- **Limit:** K is *data-independent* during inference; dynamics do not adapt to token content.



What else could be done?

Goal: Preserve S4's stability/speed, but make memory *selective*—input-aware at every step.

Why selectivity? (Linguistic intuition)

Different words require different memory actions:

- “*Not*” should quickly suppress some prior semantic commitments.
- Named entities (“*Paris*”) may need slower decay to maintain discourse topic.
- Discourse markers (“*however*”) should reweight what gets carried forward.



“John promised Mary he would not visit Paris yesterday.”

- Subject long-range link: John → promised → he
- Negation pivot: not should *selectively* attenuate “visit”-related intent
- Entity/topic: Paris should persist moderately (location context)
- Temporal cue: yesterday affects temporal interpretation briefly

We want the model to *adapt* its memory update when these tokens arrive.



From *Fixed* to *Selective* Dynamics

Idea: Let the SSM parameters depend on the current input.

$$\dot{x}(t) = A(\underline{u_t})x(t) + B(\underline{u_t})u_t, \quad y(t) = C(\underline{u_t})x(t)$$

Plain words: The “gears” of memory (decay, mixing, readout) reconfigure based on the token embedding u_t .

- For “not”: increase decay of certain semantic channels.
- For “Paris”: slow decay for entity/location channels.



Selective Parameterization (per dimension)

We use elementwise (diagonal) dynamics for speed & stability:

$$A(u_t) = -\Lambda(u_t) = -\text{diag}(\lambda_1(u_t), \dots, \lambda_n(u_t))$$

$$B(u_t) = b(u_t) \in \mathbb{R}^n, \quad C(u_t) = c(u_t) \in \mathbb{R}^n$$

Stability: ensure $\lambda_i(u_t) > 0$ via $\lambda_i(u_t) = \text{softplus}(\tilde{\lambda}_i(u_t))$.

- **Math effect:** Negative diagonal $A \Rightarrow$ exponentially stable.
- **Language effect:** Each memory “dial” has a learnable, input-controlled decay rate.



How to infuse input?

Compute gates/features from token embedding u_t :

$$g_t = W_g u_t, \quad \tilde{\lambda}_t = W_\lambda u_t, \quad b_t = W_b u_t, \quad c_t = W_c u_t$$

$$\lambda_t = \text{softplus}(\tilde{\lambda}_t), \quad A(u_t) = -\text{diag}(\lambda_t)$$

Plain words: A small network converts the token into *knobs* that set decay (forget), input injection, and readout.



Let's do the discretization now

Consider a (possibly) input-dependent step $\Delta_t > 0$.

$$x_{t+1} = \bar{A}_t x_t + \bar{B}_t u_t, \quad y_t = g_t \odot (c_t \odot x_t)$$

For diagonal $A(u_t) = -\Lambda_t$,

$$\bar{A}_t = e^{-\Lambda_t \Delta_t} \quad (\text{elementwise } e^{-\lambda_{t,i} \Delta_t})$$

$$\bar{B}_t = \left(\int_0^{\Delta_t} e^{-\Lambda_t \tau} d\tau \right) b_t = \Lambda_t^{-1} (I - e^{-\Lambda_t \Delta_t}) b_t$$

Intuition: Bigger $\lambda_{t,i}$ or step Δ_t means faster forgetting on that channel at this token.



Recurrence made faster via Gates

Let $\alpha_t := e^{-\lambda_t \odot \Delta_t} \in \mathbb{R}^n$ and $\beta_t := \frac{1 - \alpha_t}{\lambda_t} \odot b_t \in \mathbb{R}^n$ (elementwise ops).

Then:

$$x_{t+1} = \alpha_t \odot x_t + \beta_t \odot u_t, \quad y_t = c_t \odot x_t.$$

Plain words:

- α_t : how much of the old memory to keep (per channel, per token).
- β_t : how strongly to inject the current token (per channel, per token).
- c_t : which channels to read out (attention-like emphasis without pairwise scores).



Derivation of Beta (update) Gate

For one channel with $\lambda > 0$,

$$\bar{B} = \int_0^{\Delta} e^{-\lambda\tau} d\tau b = \left[\frac{-1}{\lambda} e^{-\lambda\tau} \right]_{\tau=0}^{\Delta} b = \frac{1 - e^{-\lambda\Delta}}{\lambda} b.$$

Thus $\beta = \frac{1-\alpha}{\lambda} b$ with $\alpha = e^{-\lambda\Delta}$.



Gated Recurrence – Linguistic Interpretation

- **At “not”**: λ_t spikes for channels tied to *affirmative intent*; α_t drops \Rightarrow those traces fade.
- **At “Paris”**: λ_t smaller for *entity/location* channels; α_t higher \Rightarrow topic place persists.
- **At “yesterday”**: transient temporal channel: moderate β_t (inject), then faster decay next steps.
- **Pronoun “he”**: c_t highlights the channel that binds back to John.



Why is the computation Linear-time? – *Selective Scan*

For each dimension i :

$$x_{t+1,i} = \alpha_{t,i} x_{t,i} + \beta_{t,i} u_{t,i}.$$

This is a simple first-order recurrence per channel. We *scan* it forward in time:

$$x_{t,i} = \left(\prod_{j<t} \alpha_{j,i} \right) x_{0,i} + \sum_{k<t} \left(\beta_{k,i} u_{k,i} \prod_{j=k+1}^{t-1} \alpha_{j,i} \right).$$

Implementation: Use associative scan/prefix tricks and a fused, memory-efficient kernel. Complexity $O(Tn)$ and streaming-friendly.



Linear-time *Selective Prefix (Associative) Scan*

Numerical Example:

$$\alpha = [0.9, 0.8, 0.5, 0.7], \quad \beta = [0.1, 0.2, 0.3, 0.4], \quad u = [10, 20, 30, 40]$$

t	α_t	Prefix $p_t = \prod_{j < t} \alpha_j$	Incremental Update
0	0.9	1.00	$x_1 = 0.9x_0 + 1.0$
1	0.8	0.9	$x_2 = 0.8x_1 + 4.0$
2	0.5	0.72	$x_3 = 0.5x_2 + 9.0$
3	0.7	0.36	$x_4 = 0.7x_3 + 16.0$

Each new step reuses the previous prefix p_t —no need to revisit all earlier tokens.

Linguistic Analogy:

- Reading “*John promised Mary he would not visit Paris yesterday*” —
 - α_t : how much of the prior meaning to retain,
 - $\beta_t u_t$: what new semantic cue to inject.
- Mamba just keeps one evolving summary—like updating a mental



The Mamba Kernel

S4: fixed convolution kernel K .

$$y_t = \sum_k K_k u_{t-k}, \quad K_k \text{ depends on } \bar{A}, \bar{B}, C \text{ only.}$$

Mamba: *token-conditioned* kernel \Rightarrow effectively K_k depends on u_{t-k} via (α, β, c) .

$$y_t = \sum_k \underbrace{\left[c_t \odot \left(\beta_{t-k} \odot \prod_{j=t-k+1}^{t-1} \alpha_j \right) \right]}_{\text{data-dependent weight}} \odot u_{t-k}.$$

Plain words: The filter reshapes itself on the fly based on the content it is filtering.



Ensuring Stability

- **Decay positivity:** $\lambda_t = \text{softplus}(W_\lambda u_t)$ ensures $\lambda_t > 0$.
- **Step positivity:** $\Delta_t = \text{softplus}(w_\Delta^\top u_t + b_\Delta)$.
- **Bounded keep rate:** $\alpha_t = e^{-\lambda_t \odot \Delta_t} \in (0, 1]$ avoids explosions.
- **Gradient flow:** Per-step contraction curbs exploding gradients; input-dependent α_t combats vanishing by slowing decay where needed.



Mamba Architectural Pipeline

- 1 Start with S4's stable backbone: diagonalizable A with decaying modes.
- 2 Specialize to diagonal A for fast per-dim recurrences.
- 3 Make A, B, C selective: $A(u_t) = -\text{diag}(\lambda_t)$, b_t, c_t from u_t .
- 4 Discretize: $\alpha_t = e^{-\lambda_t \odot \Delta_t}$, $\beta_t = \frac{1 - \alpha_t}{\lambda_t} \odot b_t$.
- 5 Recurrent update: $x_{t+1} = \alpha_t \odot x_t + \beta_t \odot u_t$, $y_t = c_t \odot x_t$.
- 6 Implement with a fused linear-time *selective scan*; streamable and parallel.



Performance comparison

- Evaluated on **8 zero-shot benchmarks** (Pile, LAMBADA, HellaSwag, PIQA, ARC-E, ARC-C, WinoGrande).
- Compared against open-source Transformers (GPT-NeoX, OPT, Pythia, RWKV) trained on the same data budget (300B tokens).
- **Goal:** Does a linear-time, selective SSM match or outperform quadratic-time attention?

Experimental Design

Each model family was evaluated at comparable sizes: 370M → 790M → 1.4B → 2.8B → 6.7B parameters.

Result Summary: Across all scales, **Mamba matches or exceeds Transformers**—often at *half the parameter count*.



Performance comparison

Table 2: (Zero-shot Evaluations.) Best results for each size in bold. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba is best-in-class on every single evaluation result, and generally matches baselines at twice the model size.

MODEL	TOKEN.	PILE PPL ↓	LAMBADA PPL ↓	LAMBADA ACC ↑	HELLASWAG ACC ↑	PIQA ACC ↑	ARC-E ACC ↑	ARC-C ACC ↑	WINOGRANDE ACC ↑	AVERAGE ACC ↑
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



Comparison Highlights:

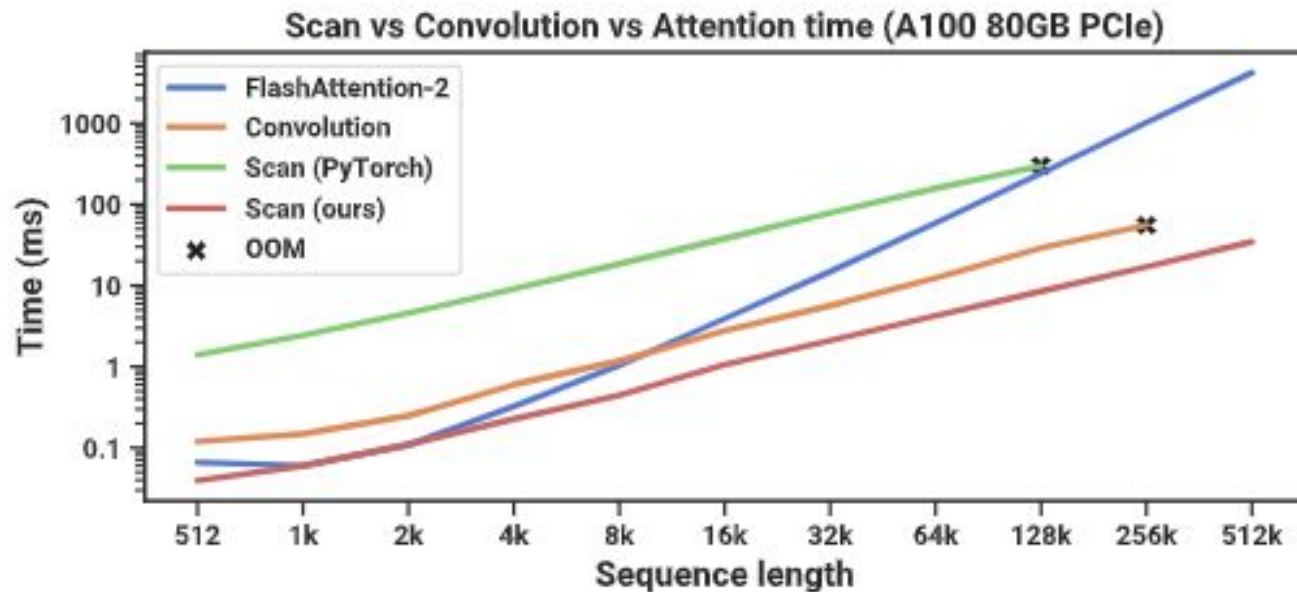
- Mamba 370M Pythia 410M and beats Hybrid H3 360M.
- Mamba 1.4B and 2.8B match Transformers nearly **twice the size**.
- Mamba 6.7B achieves **state-of-the-art zero-shot average 62.9**, comparable to

Takeaway

Efficiency: Linear-time memory mechanisms don't just scale—they generalize competitively. **Meaning:** Stable, selective recurrence can capture long-range dependencies that attention usually handles.



Training Efficiency in Context



Training Efficiency in Context

- For short sequences (< 2 k tokens), all methods are comparable.
- As sequences grow (up to 512 k), attention cost rises sharply; Mamba remains near-linear.
- FlashAttention-2 still hits out-of-memory (OOM) beyond 256 k, while Mamba trains stably.

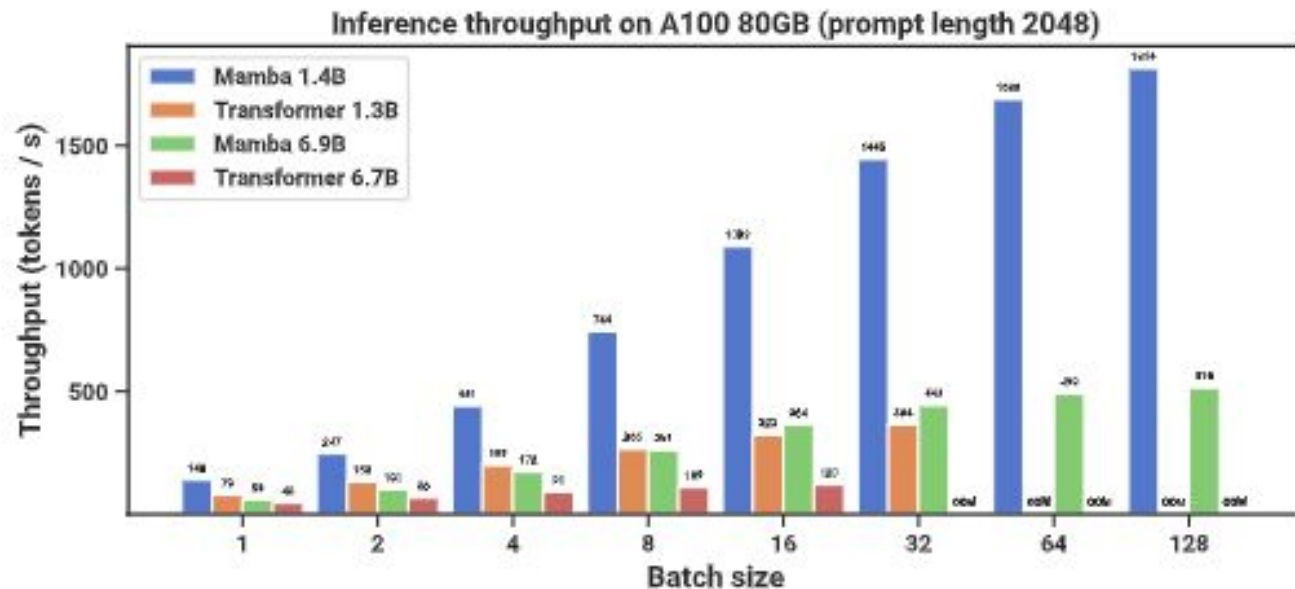
Takeaway:

Scaling Property

Training time \propto sequence length for Mamba vs \propto (sequence length)² for attention.



Inference Throughput in Context



Inference Throughput in Context

- As batch size increases, Transformer throughput saturates early (memory-bound).
- Mamba scales almost linearly with batch size: e.g., at batch 128, $\sim 5 \times$ more tokens/s.
- Both small (1.4B) and large (6.9B) Mambas outperform comparable Transformers.

Why?

No quadratic KV cache \rightarrow no exponential growth in memory. State updates are constant-size and GPU-friendly.

Linguistic intuition: Instead of recalling every prior word, Mamba “remembers what matters” through a fixed-size evolving state—hence faster inference at longer contexts.



Summarizing ...

Quantitative Highlights

- **40×** faster training than standard scan.
- **5× higher** inference throughput than Transformers.
- Linear compute–memory scaling.

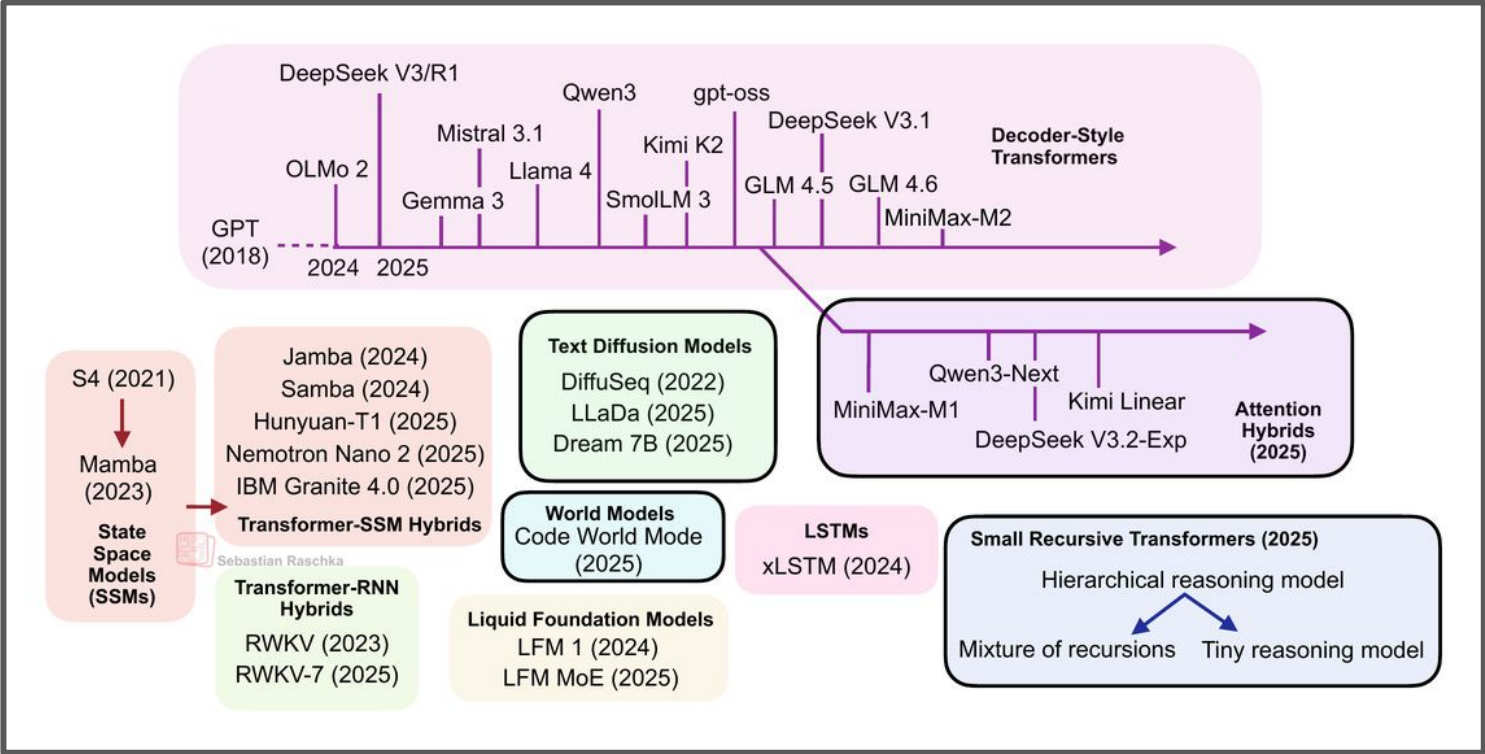
Qualitative Insights

- Efficiency from selective recurrence, not sparse tricks.
- Enables ultra-long sequences (> 256 k tokens) without OOM.
- Streaming-friendly — ideal for autoregressive tasks.

Takeaway: Mamba combines long-range reasoning with Transformer-level performance—
but runs at *linear speed and stable memory*.



Most recent developments across the landscape ...



Credits: Sebastian Raschka



Questions?

