Gated Linear Recurrence for Efficient Sequence Modeling

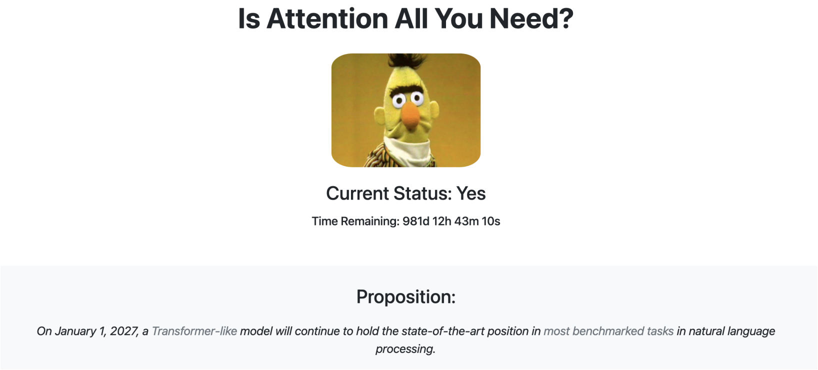
Songlin Yang

MIT

April 2024

Introduction

Is attention all you need?



Is Attention All You Need?

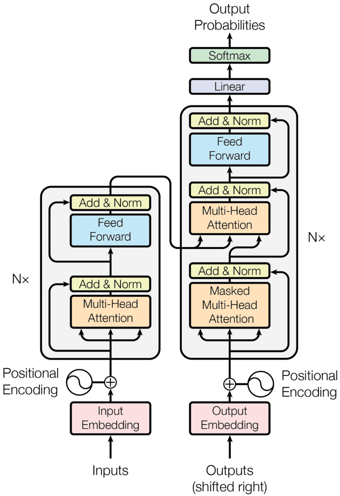
Current Status: Yes

Time Remaining: 981d 12h 43m 10s

Proposition:

On January 1, 2027, a Transformer-like model will continue to hold the state-of-the-art position in most benchmarked tasks in natural language processing.

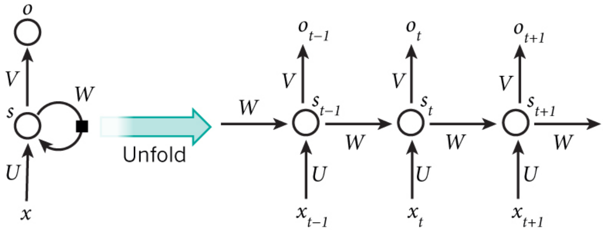
Issues with Transformers



Training: quadratic complexity
Long sequence training becomes a challenge.

Inference: KV cache grows linearly w.r.t. generation length
High memory cost

Revisiting RNNs



Training: linear complexity (but not necessarily fast on GPUs)

Inference: constant memory

Why did transformers dethrone RNNs?

- Better performance across different tasks and modalities.
- More GPU-friendly.

Research questions

How to make RNNs more performant?

How to improve RNNs' training efficiency on hardware?

Research questions

How to make RNNs more performant?

- State expansion: Increase the recurrent state size
- Selection mechanism: Better exploit fixed-size states by selectively retaining information.

Research questions

How to improve RNNs' training efficiency on hardware?

- Linearization: Linear recurrence could be parallelized in the sequence dimension.
- Structured state expansion: Outer product-based state expansion via the linear attention mechanism

Contributions

- Flash Linear Attention Library
  - I/O-aware efficient implementation for linear attention
- Incorporate selection mechanism into 2D linear RNNs
  - Gated linear attention (GLA) and HGRN2 perform comparably to LLaMa architecture transformer in language modeling.
  - Efficient chunkwise algorithm for hardware-efficient training.

Linear attention: RNNs with 2D hidden state

Recap: softmax attention

Parallel form for training:

$$\begin{aligned} Q, K, V &= XW_Q, XW_K, XW_V && \in \mathbb{R}^{L \times d} \\ O &= \operatorname{softmax}\left((QK^\intercal) \odot M\right) V && \in \mathbb{R}^{L \times d} \end{aligned}$$

Recap: softmax attention

Iterative form for inference:

$$q_t,\ k_t,\ v_t = x_t W_Q,\ x_t W_K,\ x_t W_V \qquad \in \mathbb{R}^{1\times d}$$

$$o_t = \frac{\sum_{i=1}^{t} \exp(q_t k_i^{\top}) v_i}{\sum_{i=1}^{t} \exp(q_t k_i^{\top})} = \frac{\exp(q_t K_{1:t}^{T}) V_{1:t}}{\exp(q_t K_{1:t}^{T})} \qquad \in \mathbb{R}^{1\times d}$$

Need access to all past keys and values (i.e., KV cache)

Linear attention [Katharopoulos et al., 2020]

Replace $\exp(q_t k_i^\intercal)$ with a kernel $k(x, y)$ with an associated feature map ϕ (i.e., $k(x, y) = \langle \phi(x), \phi(y) \rangle$)

$$o_t = \frac{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^\intercal v_i}{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^\intercal} = \frac{\phi(q_t)\sum_{i=1}^{t} \phi(k_i)^\intercal v_i}{\phi(q_t)\sum_{i=1}^{t} \phi(k_i)^\intercal}.$$

Prefix sum could be parallelized via the parallel scan algorithm

Linear attention [Katharopoulos et al., 2020]

$$o_t = \frac{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^\intercal v_i}{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^\intercal} = \frac{\phi(q_t) \sum_{i=1}^{t} \phi(k_i)^\intercal v_i}{\phi(q_t) \sum_{i=1}^{t} \phi(k_i)^\intercal}.$$

Letting $S_t = \sum_{i=1}^{t} \phi(k_i)^\intercal v_i$ and $z_t = \sum_{i=1}^{t} \phi(k_i)^\intercal$ where

$S_t \in \mathbb{R}^{d \times d}, z_t \in \mathbb{R}^{d \times 1}$, we can rewrite the above as an RNN,

$$\begin{aligned} S_t &= S_{t-1} + \phi(k_t)^\intercal v_t = S_{t-1} + \phi(k_t) \otimes v_t \\ z_t &= z_{t-1} + \phi(k_t)^\intercal \\ o_t &= \frac{\phi(q_t) S_t}{\phi(q_t) z_t}. \end{aligned}$$

Linear attention [Katharopoulos et al., 2020]

$$
\begin{aligned}
S_t &= S_{t-1} + \phi(k_t) \otimes v_t && \in \mathbb{R}^{d \times d} \\
z_t &= z_{t-1} + \phi(k_t)^\intercal && \in \mathbb{R}^{d \times 1} \\
o_t &= \frac{\phi(q_t) S_t}{\phi(q_t) z_t} && \in \mathbb{R}^{1 \times d}
\end{aligned}
$$

- S_t could be regarded as the 2d RNN hidden state.
- State expansion through outer product $\otimes$.
- State reduction through matrix-vector multiplication.

Linear attention [Katharopoulos et al., 2020]

$$
\begin{aligned}
S_t &= S_{t-1} + \phi(k_t) \otimes v_t && \in \mathbb{R}^{d \times d} \\
&\cancel{z_t = z_{t-1} + \phi(k_t) \qquad \in \mathbb{R}^{1 \times d}} \\
o_t &= \frac{\phi(q_t) S_t}{\cancel{\phi(q_t) z_t^\intercal}} && \in \mathbb{R}^{1 \times d}
\end{aligned}
$$

- Denominator is often discarded for stability considerations [Qin et al., 2022].
- ϕ could be simply set to the identity map [Sun et al., 2023, Mao, 2022]. We then omit ϕ for simplicity.

Linear attention training

Parallel form:

$$O = \big((QK^\intercal) \odot M\big)V$$

Due to existence of M, the training complexity is still quadratic.

Recurrent form:

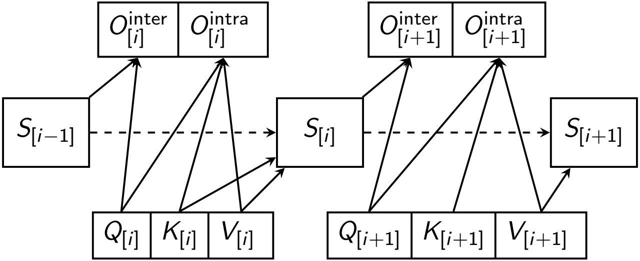
$$\begin{aligned} S_t &= S_{t-1} + k_t \otimes v_t \quad && \in \mathbb{R}^{d \times d} \\ o_t &= q_t S_t && \in \mathbb{R}^{1 \times d} \end{aligned}$$

- Lack of parallelism in the sequence dimension.
- We need to store each step's hidden state, which is memory inefficient.
    - The parallel scan algorithm is not suitable for linear attention because of this.

Chunkwise linear attention [Hua et al., 2022]

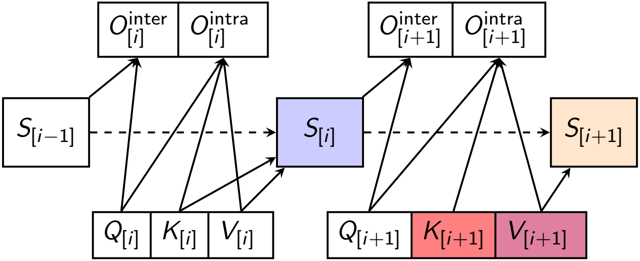
Chunk Q, K, V, O to $\{Q_{[i]}\}, \{K_{[i]}\}, \{V_{[i]}\}, \{O_{[i]}\}$, each chunk is of shape $C \times d$ where C is the chunk size

$$
\begin{aligned}
S_t &= S_{t-1} + k_t \otimes v_t \quad && \in \mathbb{R}^{d \times d} \\
o_t &= q_t S_t && \in \mathbb{R}^{1 \times d}
\end{aligned}
$$



$S_{[i]} := S_{iC}$ is the last hidden state of the i-th chunk.

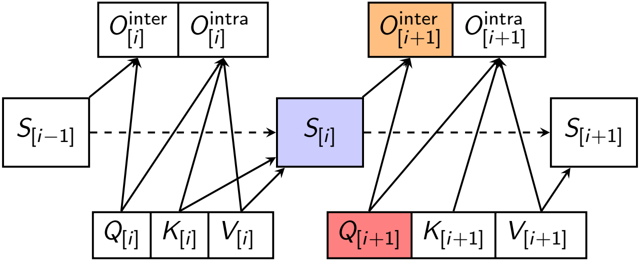
Chunkwise linear attention: Hidden state update



$$S_{[i+1]} = S_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} k_j \otimes v_j}_{K_{[i]}^{\intercal} V_{[i]}} \qquad \in \mathbb{R}^{d \times d}.$$

Aggregation of input at the chunk level through matmul in $O(Ld^2)$ time

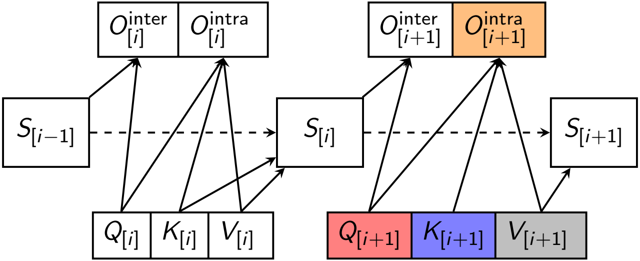
Chunkwise linear attention: inter-chunk computation of output



$$O_{[i+1]} = \underbrace{Q_{[i+1]}S_{[i]}}_{\text{inter-chunk:}\,O_{[i+1]}^{\text{inter}}} + \underbrace{\left(\left(Q_{[i+1]}K_{[i+1]}^{\mathsf{T}}\right)\odot M\right)V_{[i+1]}}_{\text{intra-chunk:}\,O_{[i+1]}^{\text{intra}}} \in \mathbb{R}^{C\times d}$$

Current chunk's query vectors attend to previous chunk's last hidden state to compute O^{inter} in $O(Ld^2)$ time

Chunkwise linear attention: intra-chunk computation of output



$$O_{[i+1]} = \underbrace{Q_{[i+1]}S_{[i]}}_{\text{inter-chunk:}O_{[i+1]}^{\text{inter}}} + \underbrace{\left(\left(Q_{[i+1]}K_{[i+1]}^{\mathsf{T}}\right) \odot M\right) V_{[i+1]}}_{\text{intra-chunk:}O_{[i+1]}^{\text{intra}}} \in \mathbb{R}^{C \times d}$$

Chunkwise self-attention-style computation of O^{intra} in $O((L/C)C^2d) = O(LCd)$ time.

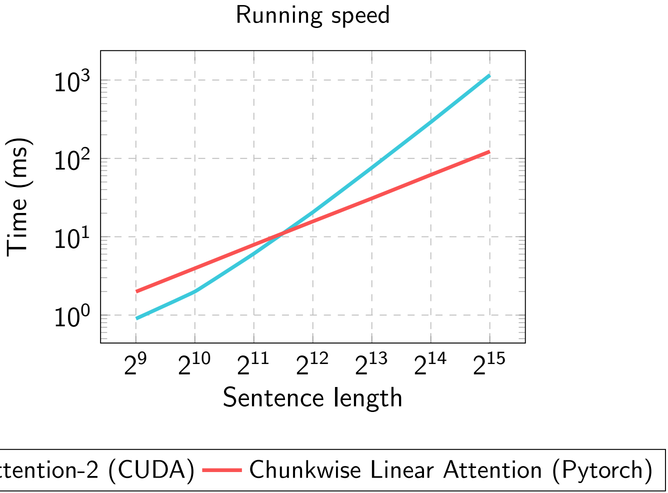
- Still linear when C is set to a small constant independent of L

Chunkwise linear attention

$$\text{The chunkwise form reduces to } \begin{cases} \text{the recurrent form}, & \text{if } C = 1 \\ \text{the parallel form}, & \text{if } C = L \end{cases}$$

Flash Linear Attention: Hardware-Efficient Linear Attention

Motivation



- For a regular 2K training length setting, chunkwise linear attention is slower than (quadractic) FlashAttention2.

Flashattention is IO-aware

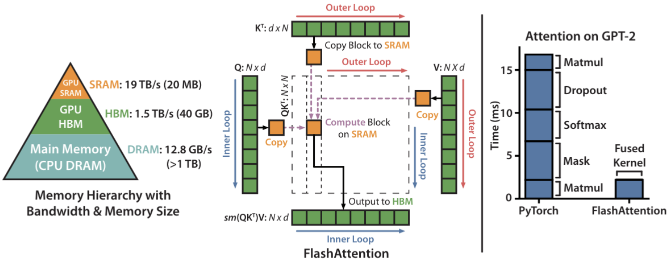
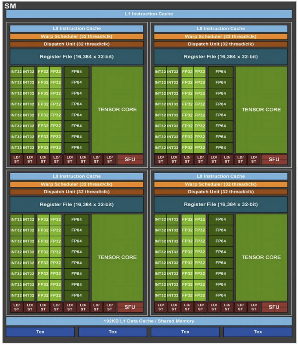


Figure: Copied from [Dao et al., 2022]

Attention operator is memory-bounded, FlashAttention reduces I/O cost by

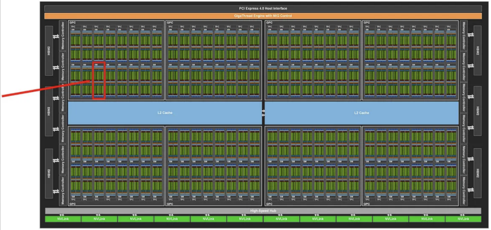
- kernel fusion
- recomputation to avoid materializing LxL attention matrix to global memory

Basic of GPU concepts: streaming multiprocessors



SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

SM

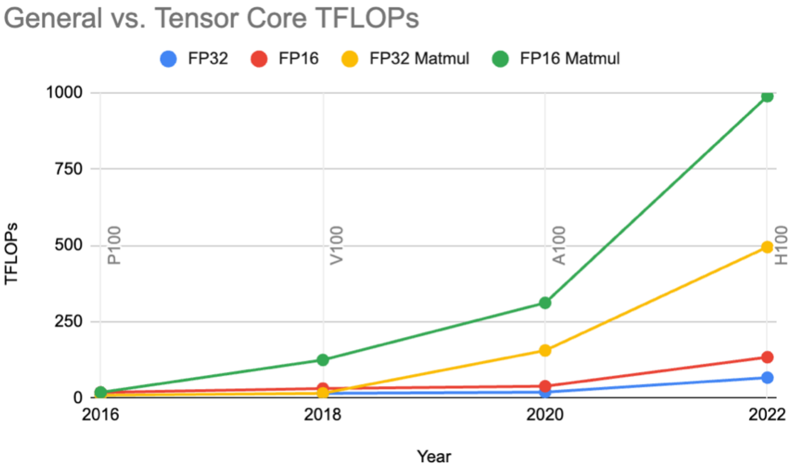


PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
Memory Controller
HBM2
High-Speed Hub
NVLink

GA100 Full GPU with 128 SMs

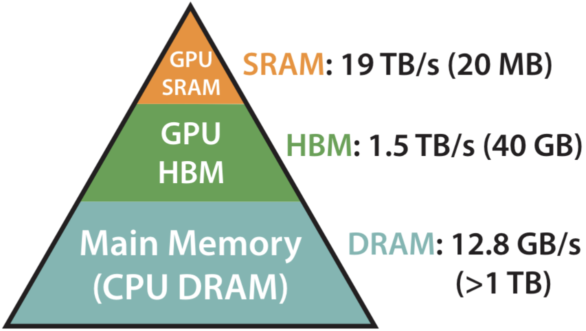
GPUs have many threads executed in parallel; threads are grouped into thread blocks, which execute on streaming multiprocessors (SMs).

Basic of GPU concepts: tensor cores



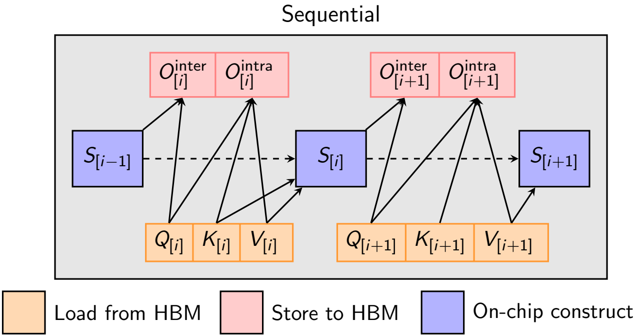
Half-prevision matrix multiple is much faster (around 16x) on GPUs

Basic of GPU concepts: memory hierarchy



Memory Hierarchy with Bandwidth & Memory Size

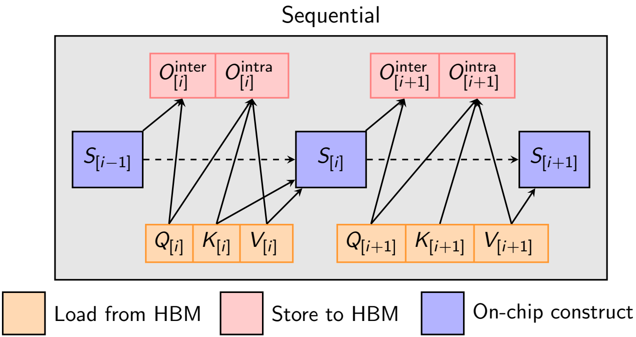
Flash linear attention (the non-materialization version)



Pros:

- S does not materialize to HBM, saving I/O cost and memory.
- $Q_{[i]}, K_{[i]}, V_{[i]}$ are loaded only once to compute both $O_{[i]}$ and $S_{[i]}$.

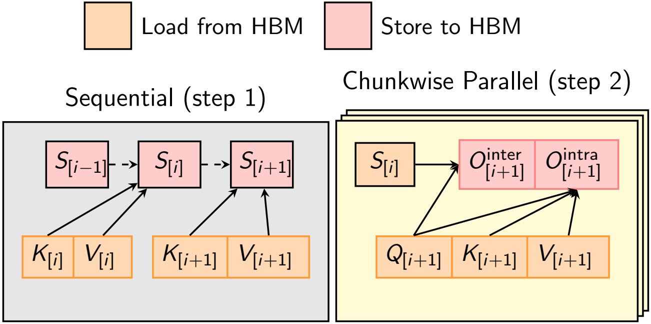
Flash linear attention (the non-materialization version)



Cons:

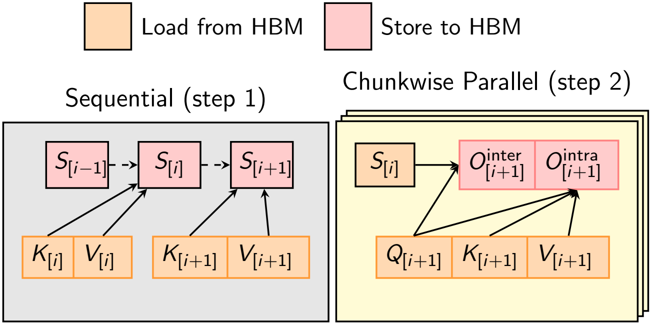
- Sequential computation, limited sequence-level parallelism.

Flash linear attention (the materialization version)



- Step1 in sequential: Compute and store hidden states.
- Step2 in parallel: For each chunk, load the previous chunk's last hidden state and compute the outputs.

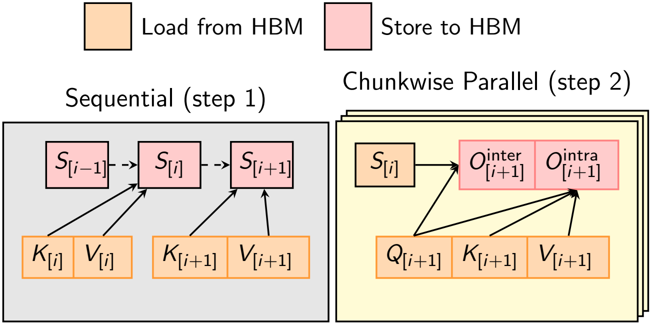
Flash linear attention (the materialization version)



Pros:

- Less FLOPs spent on recurrence. Chunkwise parallel in the second stage.

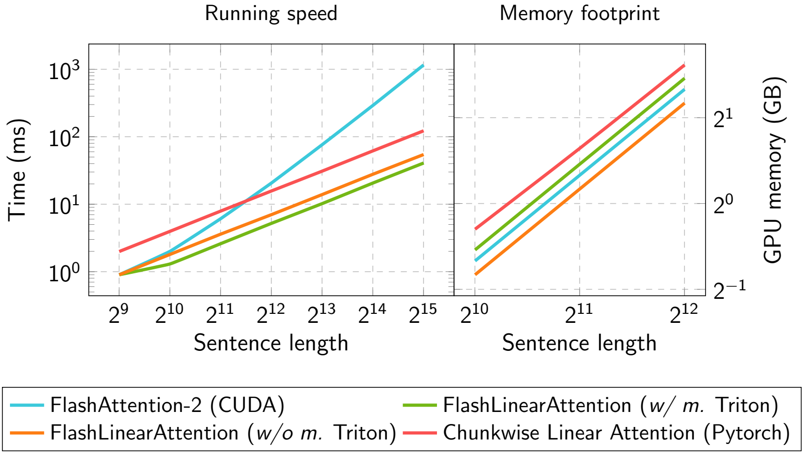
Flash linear attention (the materialization version)



Cons:

- Need to store hidden states to HBMs. More I/O costs and memory usage.

Speed comparsion



- Faster than flashattention2 even for short sequences (e.g., 1K).
- 2x-3x faster than the naive Pytorch implementation thanks to the I/O-awareness.

Flash linear attention is open-sourced!

Our goal is to provide state-of-the-art linear attention implementations under Huggingface model interface with hardware-efficient Triton kernels. Current support:

- GLA [Yang et al., 2023]
- RetNet [Sun et al., 2023]
- Based linear attention [Arora et al., 2024]
- RWKV-v6 [Peng et al., 2024]
- HGRN2 [Qin et al., 2024]
- ...

Figure: Star flash-linear-attention here if you're interested

Linear Attention with Selection Mechanism

Vanilla linear attention doesn't perform well in language modeling

Linear attention is fast, but what about the performance?

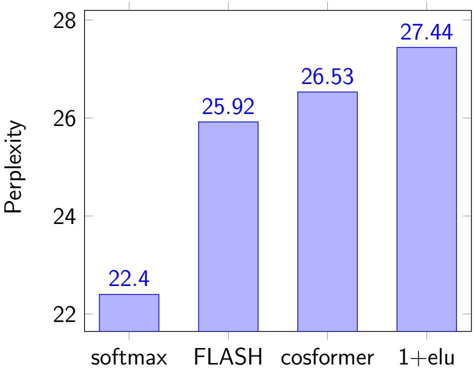


Figure: Small-scale language modeling performance on Wiki103 reported by [Qin et al., 2023].

Vanilla Linear Attention

$$S_t = I_d S_{t-1} + \phi(k_t) \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

The transition matrix $I_d \in \mathbb{R}^{d \times d}$ is the identity matrix. There is no forgetting mechanism to erase irrelevant information!

Linear attention with decay: RetNet [Sun et al., 2023]

$$S_t = \gamma I_d S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

where $\gamma \in (0, 1)$.

- Past information is decayed exponentially.
- Improved performance of language modeling.

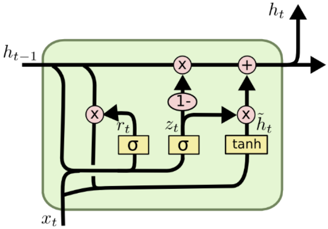
Linear attention with decay: RetNet [Sun et al., 2023]

$$S_t = \gamma I_d S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

where $\gamma \in (0, 1)$.

There is no selection mechanism to decide what information to remember and what to forget.

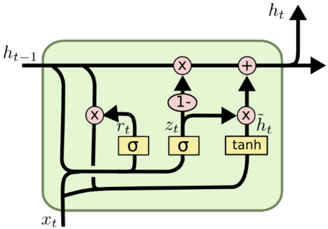
Selection mechanism example: Gated Recurrent Unit [Chung et al., 2014]



$$
\begin{aligned}
z_t &= \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right)\\
r_t &= \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right)\\
\hat{h}_t &= \phi\left(W_h x_t + U_h\left(r_t \odot h_{t-1}\right) + b_h\right)\\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

- $x_t \in \mathbb{R}^d$: input vector
- $h_t \in \mathbb{R}^d$: hidden state
- $r_t \in (0,1)^d$: update gate (i.e., forget gate)
- $z_t \in (0,1)^d$: reset gate

Why is GRU training inefficient?



$$
\begin{aligned}
z_t &= \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) \\
r_t &= \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \\
\hat{h}_t &= \phi\left(W_h x_t + U_h\left(r_t \odot h_{t-1}\right) + b_h\right) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

z_t, r_t and $\tilde{h}_t$ nonlinearly depend on h_{t-1}, which requires sequential computation.

Solution: removing state-to-state dependencies

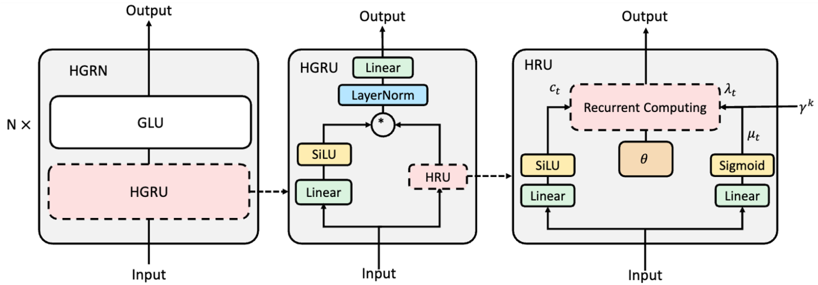
$$
\begin{aligned}
z_t &= \sigma\left(W_z x_t + \cancel{U_z h_{t-1}} + b_z\right) \\
\cancel{r_t} &= \cancel{\sigma\left(W_r x_t + U_r h_{t-1} + b_r\right)} \\
\hat{h}_t &= \phi\left(W_h x_t + \cancel{U_h\left(r_t \odot h_{t-1}\right)} + b_h\right)
\end{aligned}
$$

z and $\tilde{h}$ could be computed in parallel *before* the recurrence.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Linear recurrence could be parallelized via the parallel scan algorithm [Martin and Cundy, 2018].

HGRN [Qin*, **Yang*** et al., 2023]



- Transformer-like architecture.
- Replace self-attention layers with gated linear recurrent layers for fast token mixing.
- Channel mixing layers (i.e., GLU) can capture non-linear dependencies between different time steps.

Data-dependent decay is important for linear RNNs

Most previous linear recurrent models use data-independent decay:

- S4 [Gu et al., 2022]
- S5 [Smith et al., 2023]
- Mega [Ma et al., 2022]
- LRU [Orvieto et al., 2023]
- RWKV-v4 [Peng et al., 2023]
- RetNet [Sun et al., 2023]

Most recent linear recurrent models use data-dependent decay:

- Mamba [Gu and Dao, 2023]
- GLA [Yang et al., 2023]
- Hawk and Griffin [De et al., 2024]
- RWKV-v6 [Peng et al., 2024]
- HGRN2 [Qin et al., 2024]

Linear attention with selection mechanism

Generalize selection mechanism to the 2D RNN case:

$$S_t = A_t S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

The data-dependent transition matrices $A_t \in [0,1]^{d \times d}$ vary for each time step.

Data-dependent

$A_t = f(x_t)$ for some function f with learnable parameters

Linear attention with selection mechanism

$$S_t = A_t S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d\times d}$$

Unroll:

$$S_t = \sum_{j=1}^{t-1} \Big(\prod_{k=j+1}^{t} A_t \Big)(k_j \otimes v_t)$$

To make training efficient, it is necessary to calculate the cumulative product of the transition matrices efficiently.

Linear attention with selection mechanism

$$S_t = A_t S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

Unroll:

$$S_t = \sum_{j=1}^{t-1} \left(\prod_{k=j+1}^{t} A_t \right) (\phi(k_j) \otimes v_t)$$

However, computing the cumulative matrix product with arbitrary dense matrices A_t takes $O(Ld^3)$ time, making training expensive.

GLA [**Yang***, Wang* et al, 2023]: Gated linear attention with diagonal transition matrix

$$S_t = \mathrm{Diag}(\alpha_t) S_{t-1} + k_t \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

where $\alpha_t \in [0, 1]^d$.

Cumulative product with diagonal matrices is very efficient to compute in $O(Ld)$.

Can we just use the FlashLinearAttention out of the box?

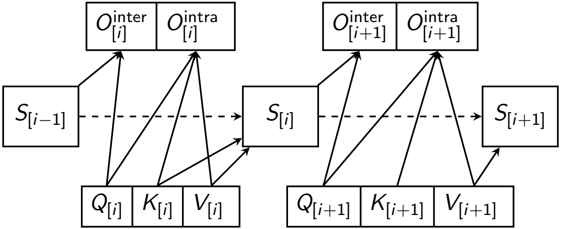
Sort of, but requires nontrivial modifications.

Chunkwise GLA training: hidden state update

Given the i-th chunk, denote

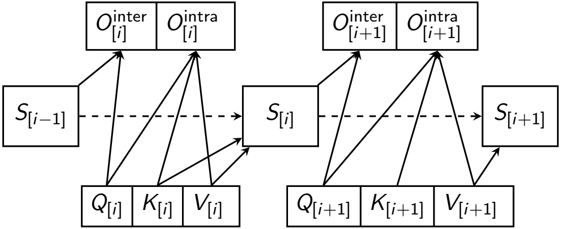
- $(B_{[i]})_j := \beta_{iC+j} = \prod_{k=iC+1}^{iC+j} \mathrm{Diag}(\alpha_k)$: cumulative product from the start of the chunk
- $\gamma_i := (\beta_{[i]})_C$: cumulative product for the entire chunk

Chunkwise GLA: hidden state update



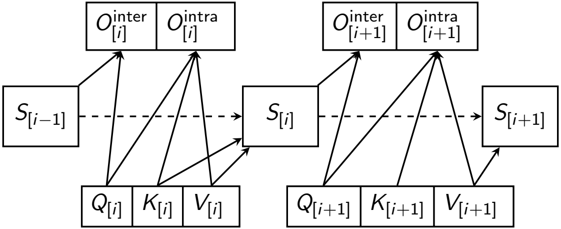
$$
\begin{aligned}
S_{[i+1]} &= \overbrace{\left(\prod_{j=iC+1}^{(i+1)C} \mathrm{Diag}(\alpha_j)\right)}^{\gamma_{i+1}} S_{[i]} \\
&\quad + \sum_{j=iC+1}^{(i+1)C} \left(\prod_{m=j}^{(i+1)C} \mathrm{Diag}(\alpha_m)\right) (k_j \otimes v_j) \qquad \in \mathbb{R}^{d \times d}
\end{aligned}
$$

Chunkwise GLA: hidden state update



$$\mathsf{S}_{[i+1]} = \left(\prod_{j=iC+1}^{(i+1)C} \mathsf{Diag}(\alpha_j)\right) S_{[i]} + \sum_{j=iC+1}^{(i+1)C} \underbrace{\left(\prod_{m=j}^{(i+1)C} \mathsf{Diag}(\alpha_m)\right) (k_j}_{(k_j/\beta_j)\gamma_{i+1}} \otimes v_j) \qquad \in \mathbb{R}^{d\times d}$$

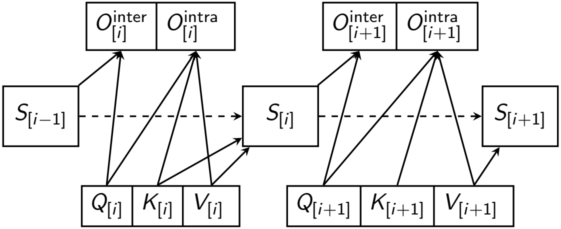
Chunkwise GLA: hidden state update



$$S_{[i+1]} = \left(\prod_{j=iC+1}^{(i+1)C} \mathrm{Diag}(\alpha_j)\right) S_{[i]} + \underbrace{\sum_{j=iC+1}^{(i+1)C} \left(\prod_{m=j}^{(i+1)C} \mathrm{Diag}(\alpha_m)\right)(k_j \otimes v_j)}_{\gamma_{i+1}(K_{[i+1]}/B_{[i+1]})^\intercal V_{[i+1]}} \qquad \in \mathbb{R}^{d\times d}$$

Efficient chunk state update via matmul

Chunkwise GLA: output computation

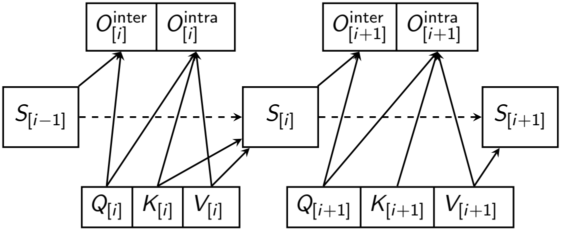


$$O^{\text{inter}}_{[i+1]} = (Q \odot B)_{[i+1]} S_{[i]} \in \mathbb{R}^{C \times d}$$

$$O^{\text{intra}}_{[i]} = \underbrace{P_{[i]}}_{\mathbb{R}^{C \times C}} V_{[i]} \in \mathbb{R}^{C \times d}$$

We could leverage tensor cores for the calculation.

Chunkwise GLA: output computation

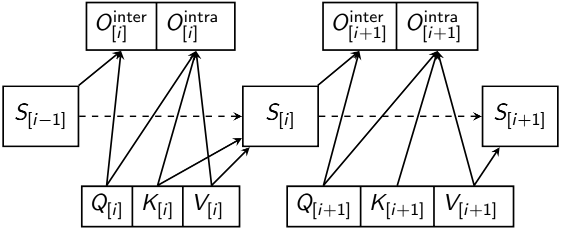


$$P_{[i]} = (Q_{[i]} \odot B_{[i]})(K_{[i]}/B_{[i]})^{\intercal}$$

😐 Numerical unstable because $B_{[i]}$ could be extremely small

- $K_{[i]}/B_{[i]}$ would explode

Chunkwise GLA: output computation

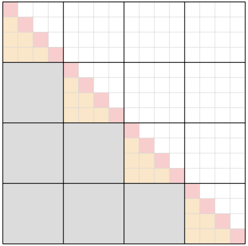


The numerical stable way to compute $P_{[i]}$ is

$$(P_{[i]})_{mn} = \begin{cases} \sum(q_{iC+m} \odot k_{iC+n} \odot \beta_{iC+m}/\beta_{iC+n}), & \text{if } m \geq n \\ 0, & \text{otherwise} \end{cases}$$

We cannot use tensor cores here

Chunkwise GLA: secondary chunking for better use of tensor cores

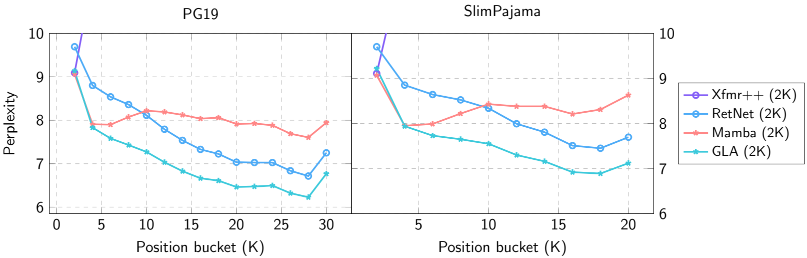


	level	tensor core
▨	1	✓
🟨	2	✓
🟥	2	
□	causal mask	

GLA experiments

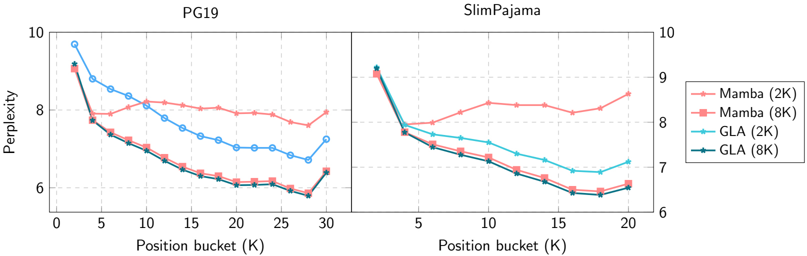
Model	Wiki.	LMB.	LMB.	PIQA	Hella.	Wino.	ARC-e	ARC-c	Avg.
	ppl ↓	ppl ↓	acc ↑	acc ↑	acc_norm ↑	acc ↑	acc ↑	acc_norm ↑	
Random baseline	-	-	-	25.0	25.0	50.0	25.0	25.0	-
340M parameters, 15B training tokens									
Transformer++	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	41.2
RetNet	32.33	49.19	28.6	63.5	33.5	52.5	44.5	23.4	41.0
Mamba	28.39	39.66	30.6	65.0	35.4	50.1	46.3	23.6	41.8
FLASH	28.83	49.84	29.4	64.6	34.4	51.1	45.9	23.5	41.5
GLA Transformer	28.65	43.35	30.3	64.8	34.5	51.4	45.1	22.7	41.5
1.3B parameters, 100B training tokens									
Transformer++	16.85	13.44	48.9	70.8	49.6	53.6	56.0	26.5	50.9
RetNet	18.64	17.27	43.3	70.0	47.3	52.5	54.8	25.6	48.9
Mamba	17.06	13.89	46.2	72.2	40.1	54.1	59.0	28.2	50.0
GLA Transformer	17.22	14.47	46.9	71.8	49.8	53.9	57.2	26.6	51.0

Length extrapolation experiment: 2K training length



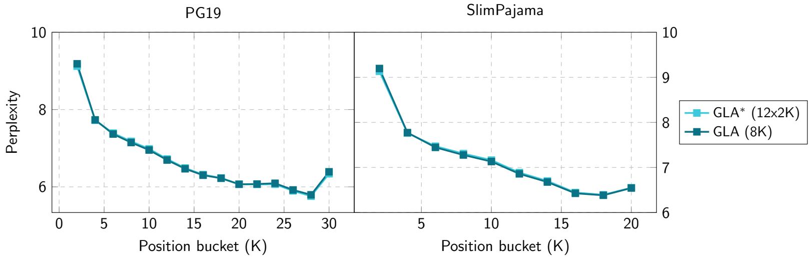
- Linear attention models (RetNet/GLA) have better length extrapolation ability than Mamba under 2K training length.
- GLA is better than RetNet in longer sequences thanks to selection mechanism.

Length extrapolation experiment: 8K training length



- Training on longer sequences can improve the language modeling performance for both Mamba and GLA.
- Mamba's length extrapolation ability is greatly improved.
  - Open question.
  - Similar observation in LongMamba.

Length extrapolation experiment: 2x12K TBPTT training



- Train on 12 consecutive chunks (2K length each) using truncated BPTT.
- Use the last hidden state of the previous chunk to initialize the next chunk's initial hidden state.
- Similar performance.
    - Possibly save communication cost between multiple GPU devices.

Training speed and memory usage

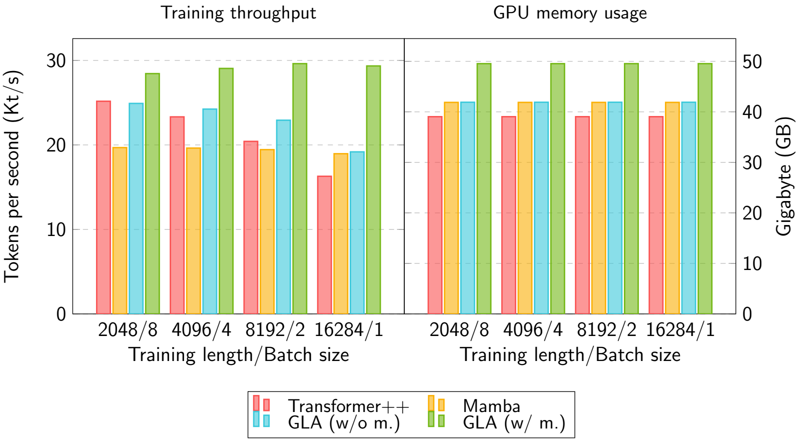


Figure: Settings: 1.3B models on a single H100.

Mamba cannot use tensor cores for linear recurrence and has limited tensor parallelism due to the use of single head.

HGRN2: GLA with RNN-inspired parameterization

GLA:

$$S_t = \mathrm{Diag}(\alpha_t) S_{t-1} + \phi(k_t) \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

$$o_t = \phi(q_i) S_t \quad \in \mathbb{R}^{d}$$

HGRN2:

$$S_t = \mathrm{Diag}(f_t) S_{t-1} + (1 - f_t) \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

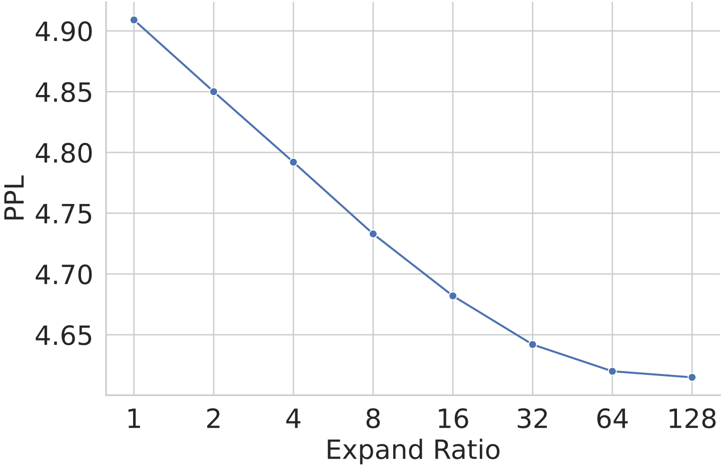
$$o_t = \tau(g_i) S_t \quad \in \mathbb{R}^{d}$$

HGRN1:

$$s_t = f_t \odot s_{t-1} + (1 - f_t) \odot v_t \in \mathbb{R}^{d}$$

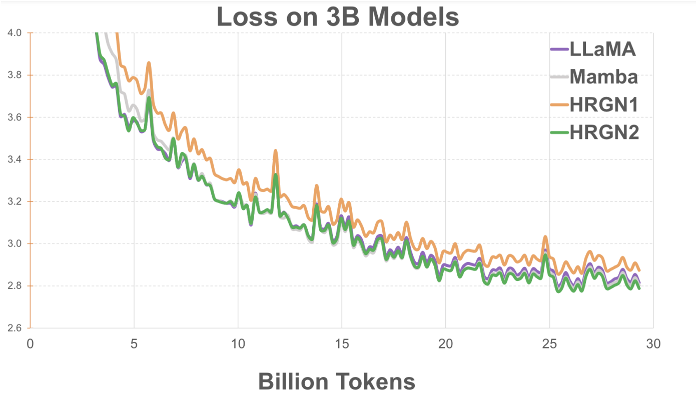
$$o_t = \tau(g_i) \odot s_t \quad \in \mathbb{R}^{d}$$

HGRN2 experiment: state expansion ablation on the Pile



Expansion ratio = head dimension of queries/keys
head dimension of HGRN1 = 1

HGRN2 experiment



Beyond diagonal transition matrix: identity plus low-rank matrix

$$S_t = (\mathrm{I} + \mathrm{w_t} \otimes \mathrm{u_t})\, S_{t-1} + \phi(k_t) \otimes v_t \quad \in \mathbb{R}^{d \times d}$$

- Cumulative matrix product could be computed efficiently via the WY representation [Bischof and Loan, 1985].
- The delta rule proposed in [Schlag et al., 2021] uses a special form of this recurrence
  - This involves the transition matrix $I - \phi(k_t) \otimes \phi(k_t)$
  - On-going research: use delta rule to improve associative recall with hardware-efficient training.

Hybrid model

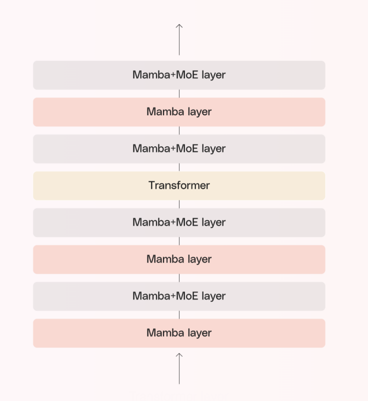


Figure: Jamba architecture

Hybrid of gated linear recurrent (Mamba) and attention layer

- Only 12.5% attention layers are needed.
- Reduce KV cache size

Hardware-efficient training of gated linear recurrent layers enables models to train on more tokens!

- LLaMa3: 15T tokens

Thank You

References I

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and Ré, C. (2024).
Simple linear attention language models balance the recall-throughput tradeoff.
CoRR, abs/2402.18668.

Bischof, C. H. and Loan, C. V. (1985).
The wy representation for products of householder matrices.
In *SIAM Conference on Parallel Processing for Scientific Computing*.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014).
Empirical evaluation of gated recurrent neural networks on sequence modeling.
CoRR, abs/1412.3555.

References II

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. (2022).
Flashattention: Fast and memory-efficient exact attention with io-awareness.
In *NeurIPS*.

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., Desjardins, G., Doucet, A., Budden, D., Teh, Y. W., Pascanu, R., de Freitas, N., and Gulcehre, C. (2024).
Griffin: Mixing gated linear recurrences with local attention for efficient language models.
ArXiv, abs/2402.19427.

Gu, A. and Dao, T. (2023).
Mamba: Linear-time sequence modeling with selective state spaces.

References III

Gu, A., Goel, K., and Ré, C. (2022).
Efficiently modeling long sequences with structured state spaces.

In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Hua, W., Dai, Z., Liu, H., and Le, Q. V. (2022).
Transformer quality in linear time.
In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S., editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 9099–9117. PMLR.

References IV

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020).
Transformers are rnns: Fast autoregressive transformers with linear attention.
In *International conference on machine learning*, pages 5156–5165. PMLR.

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., and Zettlemoyer, L. (2022).
Mega: Moving average equipped gated attention.
CoRR, abs/2209.10655.

Mao, H. H. (2022).
Fine-tuning pre-trained transformers into decaying fast weights.
In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10236–10242, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

References V

Martin, E. and Cundy, C. (2018).
Parallelizing linear recurrent neural nets over sequence length.
In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gülçehre, Ç., Pascanu, R., and De, S. (2023).
Resurrecting recurrent neural networks for long sequences.
In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 26670–26698. PMLR.

References VI

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., V., K. K. G., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R. (2023).
RWKV: reinventing rnns for the transformer era.
CoRR, abs/2305.13048.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Ferdinan, T., Hou, H., I aw Kazienko, P., Kranthikiran, G., Koco'n, J., Koptyra, B., Krishna, S., McClelland, R., Muennighoff, N., Obeid, F., Saito, A., Song, G., Tu, H., Wo'zniak, S., Zhang, R., Zhao, B., Zhao, Q., Zhou, P., Zhu, J., and Zhu, R.-J. (2024).
Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence.

References VII

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. (2022).
The devil in linear transformer.
arXiv preprint arXiv:2210.10340.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. (2024).
Hgrn2: Gated linear rnns with state expansion.

Qin, Z., Yang, S., and Zhong, Y. (2023).
Hierarchically gated recurrent neural network for sequence modeling.
CoRR, abs/2311.04823.

References VIII

Schlag, I., Irie, K., and Schmidhuber, J. (2021).
Linear transformers are secretly fast weight programmers.
In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9355–9366. PMLR.

Smith, J. T. H., Warrington, A., and Linderman, S. W. (2023).
Simplified state space layers for sequence modeling.
In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. (2023).
Retentive network: A successor to transformer for large language models.
arXiv preprint arXiv:2307.08621.

References IX

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. (2023).
Gated linear attention transformers with hardware-efficient training.
CoRR, abs/2312.06635.