Gated Linear Attention Transformers with Hardware-Efficient Training

Songlin Yang [1 *] **Bailin Wang** [1 *] **Yikang Shen** [2] **Rameswar Panda** [2] **Yoon Kim** [1]

Abstract

Transformers with linear attention allow for efficient parallel training but can simultaneously be formulated as an RNN with 2D (matrix-valued) hidden states, thus enjoying linear-time inference complexity. However, linear attention generally underperforms ordinary softmax attention. Moreover, current implementations of linear attention lack I/O-awareness and are thus slower than highly optimized implementations of softmax attention. This work describes a hardware-efficient algorithm for linear attention that trades off memory movement against parallelizability. The resulting implementation, dubbed FLASHLINEARATTENTION, is faster than FLASHATTENTION-2 (Dao, 2023) as a standalone layer even on short sequence lengths (e.g., 1K). We then generalize this algorithm to a more expressive variant of linear attention with data-dependent gates. When used as a replacement for the standard attention layer in Transformers, the resulting gated linear attention (GLA) Transformer is found to perform competitively against the LLaMA-architecture Transformer (Touvron et al., 2023) as well recent linear-time-inference baselines such as RetNet (Sun et al., 2023a) and Mamba (Gu & Dao, 2023) on moderate-scale language modeling experiments. GLA Transformer is especially effective at length generalization, enabling a model trained on 2K to generalize to sequences longer than 20K without significant perplexity degradations. For training speed, the GLA Transformer has higher throughput than a similarly-sized Mamba model.

https://github.com/sustcsonglin/flash-linear-attention

[*]Equal contribution [1]Massachusetts Institute of Technology [2]MIT-IBM Watson AI Lab. Correspondence to: Songlin Yang <yangsl66@mit.edu>, Bailin Wang <bailinw@mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1 Introduction

Transformers with softmax attention (Vaswani et al., 2017) enjoy efficient parallel training but suffer from quadratic (in sequence length) complexity, thus motivating more RNN-like models that allow for linear-time sequence modeling. Linear attention, which replaces the exponential similarity function with a simple dot product over (possibly transformed) key/query vectors, has emerged as a promising alternative to classic softmax attention (Katharopoulos et al., 2020; Choromanski et al., 2021; Kasai et al., 2021; Peng et al., 2021). An attractive property of linear attention is that it admits a "recurrent form" in which it can be formulated as a linear RNN with 2D hidden states (Katharopoulos et al., 2020), thus enabling linear-time inference. For training, linear attention also admits a subquadratic "chunkwise parallel form" which divides the sequence into non-overlapping chunks and performs (serial) inter-chunk recurrent computations followed by (parallel) intra-chunk computations (Hua et al., 2022; Sun et al., 2023a; Lingle, 2023), thus (partially) maintaining parallel training. However, existing algorithms for linear attention are not I/O aware and thus, in practice, slower than optimized implementations of softmax attention (Dao et al., 2022b; Dao, 2023) on moderate sequence lengths.

From a performance standpoint, linear attention has generally been found to underperform ordinary softmax attention, often by a significant margin in language modeling (Kasai et al., 2021). Recent variants of linear attention such as RetNet (Sun et al., 2023a) and TransNormerLLM (Qin et al., 2023b) obtain significant improvements by multiplying the current hidden state with a decay factor before the RNN update. However, these works use a global, *data-independent* decay factor, despite the fact that in 1D RNNs, a *data-dependent* gating mechanism has been shown to be crucial for performance (van der Westhuizen & Lasenby, 2018; Qin et al., 2023c). And even with the decay factor, linear attention Transformers underperform the strongest Transformer architectures when pretrained from scratch.

This work develops a hardware-efficient algorithm for linear attention, and applies it to train a gated variant of linear attention that is competitive with softmax attention. We first discuss aspects of optimizing ordinary linear attention on modern GPUs and give two I/O-aware algorithms (tailored for different training settings) based on these principles (§3). Our implementation of the algorithm, called FLASHLIN-

EARATTENTION, is faster than FLASHATTENTION-2 (Dao, 2023) even on short (e.g., 1K) sequences. We then describe a gated linear attention layer with a data-dependent gating mechanism and show how FLASHLINEARATTENTION can be generalized to the gated case (§4). We study the resulting *gated linear attention (GLA) Transformer* on moderate-scale language modeling benchmarks, where we train models with 340M/1.3B parameters on 15B/100B tokens, respectively. We find that the GLA Transformer performs favorably against a strong LLaMA architecture Transformer baseline that makes use of recent recipes (Transformer++; Touvron et al., 2023) as well as recent linear-time sequence models such as RetNet (Sun et al., 2023a) and Mamba (Gu & Dao, 2023). GLA Transformer is found to be particularly strong at length generalization and recall-intensive tasks among linear recurrent models. For training speed, the GLA Transformer has significantly higher throughput than a similarly sized Mamba model.

2 Background: Linear Attention

We first give a brief background on linear attention layers. For notation we use bold upper-case letters for matrices (e.g., $\mathbf{S}$, $\mathbf{Q}$), bold lower-case letters for vectors (e.g., $\boldsymbol{q}_t$, $\boldsymbol{k}_t$), and italic upper-case for learnable parameters matrices (e.g., $\boldsymbol{W}_K$). We generally use the same alphabet to show the rows of a matrix, e.g., $\boldsymbol{q}_t$ is the t-th row of $\mathbf{Q}$.

2.1 Parallel and Recurrent Forms

Standard autoregressive Transformers employ a softmax attention mechanism which takes an input sequence $\mathbf{X}\in\mathbb{R}^{L\times d}$ (here L is the length and d is the hidden dimension) and computes the output $\mathbf{O}\in\mathbb{R}^{L\times d}$ through,

$$\mathbf{Q},\mathbf{K},\mathbf{V}=\mathbf{X}\boldsymbol{W}_Q,\mathbf{X}\boldsymbol{W}_K,\mathbf{X}\boldsymbol{W}_V,$$
$$\mathbf{O}=\mathrm{softmax}\big((\mathbf{Q}\mathbf{K}^\top)\odot\mathbf{M}\big)\mathbf{V},$$

where $\boldsymbol{W}_Q,\boldsymbol{W}_K,\boldsymbol{W}_V\in\mathbb{R}^{d\times d}$ are learnable matrices and $\mathbf{M}\in\{-\infty,1\}^{L\times L}$ is a mask that prevents the model from attending to future tokens, i.e., $\mathbf{M}_{ij}=1$ if $i\geq j$ and $\mathbf{M}_{ij}=-\infty$ if $i<j$. (Here we assume a single attention head for simplicity.) The above *parallel form* of attention can compute $\mathbf{O}$ in parallel given the full input $\mathbf{X}$, thus enabling efficient training. However, during inference Transformers must use the following *recurrent form*,

$$\boldsymbol{q}_t,\boldsymbol{k}_t,\boldsymbol{v}_t=\boldsymbol{x}_t\boldsymbol{W}_Q,\boldsymbol{x}_t\boldsymbol{W}_K,\boldsymbol{x}_t\boldsymbol{W}_V,$$
$$\boldsymbol{o}_t=\frac{\sum_{i=1}^t\exp(\boldsymbol{q}_t\boldsymbol{k}_i^\top)\boldsymbol{v}_i}{\sum_{i=1}^t\exp(\boldsymbol{q}_t\boldsymbol{k}_i^\top)},$$

which calculates the query ($\boldsymbol{q}_t$), key ($\boldsymbol{k}_t$), and value ($\boldsymbol{v}_t$) vectors given the current token's representation $\boldsymbol{x}_t\in\mathbb{R}^{1\times d}$ and the performs attention over the (growing) set of keys $\{\boldsymbol{k}_1,...,\boldsymbol{k}_t\}$ and values $\{\boldsymbol{v}_1,...,\boldsymbol{v}_t\}$ (i.e., the "KV cache").

Linear attention mechanisms (Katharopoulos et al., 2020) replace $\exp(\boldsymbol{q}_t\boldsymbol{k}_i^\top)$ with a kernel $k(\boldsymbol{x},\boldsymbol{y})$ with an associated feature map ϕ (i.e., $k(\boldsymbol{x},\boldsymbol{y})=\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{y})\rangle$). This simplifies the calculation of $\boldsymbol{o}_t$ since we have

$$\boldsymbol{o}_t=\frac{\sum_{i=1}^t\phi(\boldsymbol{q}_t)\phi(\boldsymbol{k}_i)^\top\boldsymbol{v}_i}{\sum_{i=1}^t\phi(\boldsymbol{q}_t)\phi(\boldsymbol{k}_i)^\top}=\frac{\phi(\boldsymbol{q}_t)\sum_{i=1}^t\phi(\boldsymbol{k}_i)^\top\boldsymbol{v}_i}{\phi(\boldsymbol{q}_t)\sum_{i=1}^t\phi(\boldsymbol{k}_i)^\top}.$$

Letting $\mathbf{S}_t=\sum_{i=1}^t\phi(\boldsymbol{k}_i)^\top\boldsymbol{v}_i$ and $\boldsymbol{z}_t=\sum_{i=1}^t\phi(\boldsymbol{k}_i)^\top$ where $\mathbf{S}_t\in\mathbb{R}^{d\times d},\boldsymbol{z}_t\in\mathbb{R}^{d\times 1}$, we can rewrite the above as an RNN,

$$\mathbf{S}_t=\mathbf{S}_{t-1}+\phi(\boldsymbol{k}_t)^\top\boldsymbol{v}_t,\ \boldsymbol{z}_t=\boldsymbol{z}_{t-1}+\phi(\boldsymbol{k}_t)^\top,\ \boldsymbol{o}_t=\frac{\phi(\boldsymbol{q}_t)\mathbf{S}_t}{\phi(\boldsymbol{q}_t)\boldsymbol{z}_t}.$$

Although various kernels have been explored (Kasai et al., 2021; Peng et al., 2021), recent work has found that a linear kernel (i.e., setting ϕ to be the identity) without a normalizer works well in practice (Sun et al., 2023a). This results in an (unnormalized) linear attention layer with the following update equation,

$$\mathbf{S}_t=\mathbf{S}_{t-1}+\boldsymbol{k}_t^\top\boldsymbol{v}_t,\quad \boldsymbol{o}_t=\boldsymbol{q}_t\mathbf{S}_t. \tag{1}$$

Eq. 1 makes it clear that a linear attention layer is essentially a linear recurrent layer with matrix-valued hidden states $\mathbf{S}_t$ that is updated via the outer-product $\boldsymbol{k}_t^\top\boldsymbol{v}_t=(\boldsymbol{x}_t\boldsymbol{W}_K)^\top(\boldsymbol{x}_t\boldsymbol{W}_V)$.[1] The parallel form of causal linear attention, whose complexity is still quadratic in L, is given by $\mathbf{O}=\big((\mathbf{Q}\mathbf{K}^\top)\odot\mathbf{M}\big)\mathbf{V}$, where $\mathbf{M}\in\{0,1\}^{L\times L}$ is a mask such that $\mathbf{M}_{ij}=1$ if $i\geq j$ and $\mathbf{M}_{ij}=0$ if $i<j$. Due to $\mathbf{M}$ it is not possible to exploit the associative property of matrix multiplication to reduce the parallel form complexity from quadratic to linear.[2]

2.2 Chunkwise Parallel Form

The *chunkwise* parallel form of linear attention strikes a balance between parallel and recurrent form (Hua et al., 2022; Sun et al., 2023a), and allows for subquadratic, partially parallel training. Formally, suppose the input $\mathbf{X}$ is now split into non-overlapping chunks, where each chunk is of length C. Let $\mathbf{S}_{[i]}\in\mathbb{R}^{d\times d}$ be the chunk-level hidden state after processing i chunks, i.e., $\mathbf{S}_{[i]}:=\mathbf{S}_{iC}$. Further let $\mathbf{Q}_{[i]}:=\mathbf{Q}_{iC+1:(i+1)C+1}\in\mathbb{R}^{C\times d}$ be the query vectors corresponding to the i-th chunk; let $\mathbf{K}_{[i]}$, $\mathbf{V}_{[i]}$, $\mathbf{O}_{[i]}$ be similarly defined. We then have the following inter-chunk recurrence (for $i\in[0,1,...\frac{L}{C}-1]$):

$$\mathbf{S}_{[i+1]}=\mathbf{S}_{[i]}+\underbrace{\sum_{j=iC+1}^{(i+1)C}\boldsymbol{k}_j^\top\boldsymbol{v}_j}_{\mathbf{K}_{[i]}^\top\mathbf{V}_{[i]}}\quad\in\mathbb{R}^{d\times d}. \tag{2}$$

Here $\mathbf{S}_{[0]}$ can be initialized to zero or from the previous segment's hidden state. The sum of all RNN inputs from a chunk (i.e., $\mathbf{K}_{[i]}^\top\mathbf{V}_{[i]}$) can be computed in $O(C^2d)$ in parallel. The

[1] This type of model with matrix-valued hidden states that change over time is also known as "fast weights" (Hinton & Plaut, 1987; Schmidhuber, 1992; Ba et al., 2016), whose connection to Transformers was explored in recent work (Schlag et al., 2021; Irie et al., 2021; Mao, 2022).

[2] Without $\mathbf{M}$, one can transform $(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$ to $\mathbf{Q}(\mathbf{K}^\top\mathbf{V})$ reducing the complexity from quadratic ($O(L^2d)$) to linear ($O(Ld^2)$).

intra-chunk parallel computation for the output is given by

$$\mathbf{O}_{[i+1]} = \underbrace{\mathbf{Q}_{[i+1]}\mathbf{S}_{[i]}}_{\text{inter-chunk:}\mathbf{O}^{\text{inter}}_{[i+1]}} + \underbrace{\left((\mathbf{Q}_{[i+1]}\mathbf{K}^{\top}_{[i+1]})\odot\mathbf{M}\right)\mathbf{V}_{[i+1]}}_{\text{intra-chunk:}\mathbf{O}^{\text{intra}}_{[i+1]}},$$

where $\mathbf{O}_{[i+1]} \in \mathbb{R}^{C\times d}$. Here the "intra-chunk" component $\mathbf{O}^{\text{intra}}_{[i+1]}$ has exactly the same parallel form as Eq. 1 and thus takes $O(C^2d+Cd^2)$. The "inter-chunk" component $\mathbf{O}^{\text{inter}}_{[i+1]}$ accounts for the contribution from the hidden state from the previous chunk, and takes $O(Cd^2)$. Training complexity is thus $O\left(\frac{L}{C}(C^2d+Cd^2)\right) = O(LCd+Ld^2)$, which is less than $O(L^2d)$ when $L>d$. Note that setting $C=L$ recovers the parallel form, and $C=1$ recovers the recurrent form.

3 Hardware-Efficient Linear Attention

We describe FLASHLINEARATTENTION, an I/O-aware, hardware-efficient algorithm for linear attention in the spirit of FLASHATTENTION (Dao et al., 2022b; Dao, 2023). We first discuss aspects of hardware that should be taken into account for a practically efficient implementation.

3.1 Principles of Hardware-Efficient Algorithms

An efficient algorithm should be aware of the compute model, memory hierarchy, and specialized compute units on modern hardware.

Occupancy. GPUs have many threads executed in parallel; threads are grouped into thread blocks, which execute on streaming multiprocessors (SMs). To maintain a high GPU occupancy (i.e., fraction of GPU resources being used), it is necessary to use a sufficient number of SMs. In large-scale training and long-sequence modeling scenarios where the batch size tends to be small, parallelizing over the temporal dimension enables high GPU occupancy (Dao, 2023).

Specialized compute units. Modern hardware for neural network training typically have specialized compute units (e.g., tensor cores on NVIDIA GPUs, matrix mutiply units on TPUs), which can significantly accelerate matmuls; for example half-precision matmuls on an A100 can be roughly 16 times faster on tensor cores than on CUDA cores. These specialized units are crucial for large-scale training.

Memory hierarchy. GPUs have a memory hierarchy with larger but slower global GPU memory (high bandwidth memory; HBM) and smaller but faster shared memory (SRAM). Optimal utilization of SRAM to reduce HBM I/O cost can therefore lead to significant speed-ups.

3.2 Hardware Considerations for Linear Attention

We now discuss hardware considerations pertaining to the efficiency of the different forms of linear attention.

Recurrent form. A basic implementation of the recurrent form stores the 2D hidden states of all time steps in HBM, resulting in high I/O cost (Mao, 2022). I/O cost could be reduced by avoiding such materialization and recomputing the hidden states during the backward pass, as in Katharopoulos et al. (2020), but the elementwise operations in the recurrent update cannot make use of tensor cores and result in low arithmetic intensity. Hence, while the recurrent form generally has the lowest total FLOPs among the three forms, this does not translate to actual wall-time efficiency. And while it is theoretically possible to parallelize linear recurrences via the parallel scan algorithm, this method requires materializing the 2D hidden state for each time step. This incurs a significant memory I/O burden, thereby offsetting the benefits of parallelism over the sequence length and resulting in slow actual running speeds, as in Katsch (2023).

Algorithm 1 FLASHLINEARATTENTION: Forward Pass

Input: $\mathbf{Q},\mathbf{K},\mathbf{V} \in \mathbb{R}^{L\times d}$, $\mathbf{V} \in \mathbb{R}^{L\times d}$, chunk size $C \in [L]$, `materialize` $\in$ {`True`,`False`}
Divide $\mathbf{Q},\mathbf{K},\mathbf{V}$ into $N=\frac{L}{C}$ blocks $\{\mathbf{Q}_{[1]}\dots\mathbf{Q}_{[N]}\}$, $\{\mathbf{K}_{[1]}\dots\mathbf{K}_{[N]}\}$ of size $C\times d$ each.
Initialize $\mathbf{S}=\mathbf{0}\in\mathbb{R}^{d\times d}$ on SRAM
On chip, construct causal mask $\mathbf{M}\in\mathbb{R}^{C\times C}$
if `materialize` **then** ▷ the materialization version
  for $n \leftarrow 1,N$ **do**
    Store $\mathbf{S}$ to HBM as $\mathbf{S}_{[n]}$.
    Load $\mathbf{K}_{[n]},\mathbf{V}_{[n]}\in\mathbb{R}^{C\times d}$ from HBM to SRAM
    On chip, compute $\mathbf{S}=\mathbf{S}+\mathbf{K}^{\top}_{[n]}\mathbf{V}_{[n]}$.
  end for
  parfor $n \leftarrow 1,N$ **do**
    Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{V}_{[n]},\mathbf{S}_{[n]}$ from HBM to SRAM.
    On chip, compute $\mathbf{O}'=\mathbf{Q}_{[n]}\mathbf{S}_{[n]}+(\mathbf{Q}_{[n]}\mathbf{K}^{\top}_{[n]}\odot\mathbf{M})\mathbf{V}_{[n]}$
    Store $\mathbf{O}'$ to HBM as $\mathbf{O}_{[n]}$.
  end parfor
  return $\mathbf{O}=\{\mathbf{O}_{[1]}\dots\mathbf{O}_{[N]}\}, \mathbf{S}=\{\mathbf{S}_{[1]}\dots\mathbf{S}_{[N]}\}$.
else ▷ the non-materialization version
  for $n \leftarrow 1,N$ **do**
    Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{V}_{[n]}\in\mathbb{R}^{C\times d}$ from HBM to SRAM
    On chip, compute $\mathbf{O}'=\mathbf{Q}_{[n]}\mathbf{S}+(\mathbf{Q}_{[n]}\mathbf{K}^{\top}_{[n]}\odot\mathbf{M})\mathbf{V}_{[n]}$
    On chip, compute $\mathbf{S}=\mathbf{S}+\mathbf{K}^{\top}_{[n]}\mathbf{V}_{[n]}$.
    Store $\mathbf{O}'$ to HBM as $\mathbf{O}_{[n]}$
  end for
  return $\mathbf{O}=\{\mathbf{O}_{[1]}\dots\mathbf{O}_{[N]}\}$
end if

Parallel form. The parallel form could be as efficient as FLASHATTENTION using similar I/O optimization techniques, as demonstrated by Qin et al. (2023b). However, the high number of FLOPs (due to the quadratic complexity) makes the long-sequence training expensive, the same issue that the naïve implementation of softmax attention would suffer from.

Chunkwise form. The chunkwise parallel form, which interpolates between the parallel and recurrent forms with an extra "parameter" C, makes it possible to more easily make the above tradeoffs for fine-grained optimization. Unlike the recurrent form, most operations can be done via matmuls, enabling the use of tensor cores (if C is set to a multiple of 16). Though the chunkwise training algorithm has been discussed before in the literature (Hua et al., 2022; Sun et al., 2023a), most implementations are not I/O-aware and thus slower than FLASHATTENTION for moderate sequence lengths (e.g., 2K-4K).

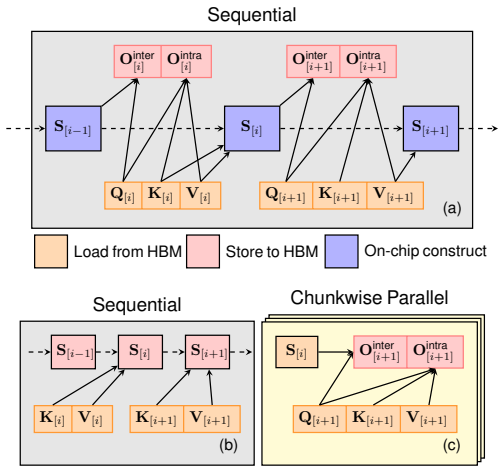

Figure 1: (a) FLASHLINEARATTENTION without materialization. This version is more memory-efficient. (b-c) FLASHLINEARATTENTION with materialization. This version enables sequence-level chunkwise parallelism.

3.3 FLASHLINEARATTENTION: Hardware-Efficient Linear Attention with the Chunkwise Form

We describe our I/O-aware, hardware-efficient implementation of the chunkwise form. We give two versions, whose forward and backward passes differ depending on whether the chunk-level hidden states $\mathbf{S}_{[n]}$ are materialized in HBM. See Alg. 1 and Fig. 1 for the forward pass. (Alg. 2 in the appendix describes the backward pass.) At a high level, we use tiling to load tensors block-by-block and re-use tensor blocks on chip to avoid multiple HBM I/O as much as possible. For example, when $\mathbf{Q}_{[n]}$ is loaded to SRAM, both $\mathbf{Q}_{[n]}\mathbf{S}$ and $(\mathbf{Q}_{[n]}\mathbf{K}_{[n]}^\top \odot \mathbf{M})\mathbf{V}_{[n]}$ can be computed on chip, which avoids loading $\mathbf{Q}_{[n]}$ twice, thus saving HBM I/O.

The non-materialization version computes $\mathbf{O}_{[n]}$ sequentially for $n \in [N]$, using SRAM to temporarily store $\mathbf{S}_{[n]}$, which is memory-efficient. This version parallelizes across batch size, number of heads, and head dimensions, but lacks sequence-level parallelim. When the batch size is large, this level of parallelism is sufficient to enable high GPU occupancy. In long-sequence and large scale training settings where batch size is small, the SMs cannot be fully exploited in this case. **The materialization version** first performs the inter-chunk recurrence (Eq. 2) and stores all $\mathbf{S}_{[n]}$ for $n \in [N]$ in HBM. Then, the $\mathbf{O}_{[n]}$'s can be computed in parallel for all chunks. This approach offers better parallelism but increases the memory footprint by approximately 10-20%. We mitigate this through *recomputation*, where the hidden states discarded after the forward pass and recomputed during the backward pass. We find this introduces a small runtime overhead but significantly reduces the memory footprint, and we adopt this strategy by default.

Figure 2 shows the speed and memory footprint of our implementation. Both versions of FLASHLINEARATTENTION are substantially faster than FLASHATTENTION-2 (Dao, 2023) and a pure PyTorch (i.e., I/O-unaware) implementation of chunkwise linear attention, showing the benefits of I/O-awareness.

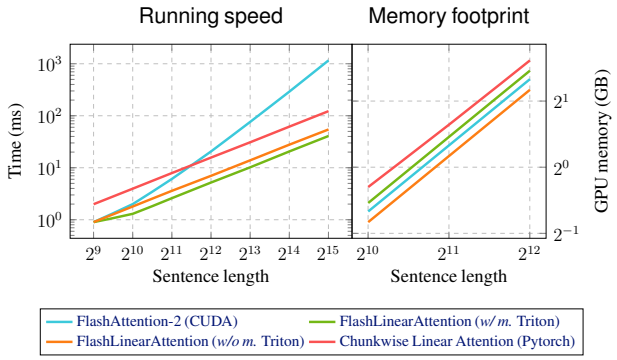

Figure 2: Speed comparison on a single H100 GPU with batch size 32, number of heads 16, head dimension 64, and chunk size 64. Both x- and y-axes are on log scale. *w/ m.* and *w/o m.* denotes using FLASHLINEARATTENTION *with* or *without materialization* of hidden states in HBM.

4 Gated Linear Attention

The linear recurrence in Eq. 1 does not have a decay term or a forget gate, which has been shown to be crucial in RNNs (Hochreiter & Schmidhuber, 1997; Cho et al., 2014; van der Westhuizen & Lasenby, 2018). The lack of a decay term makes it difficult for a model to "forget" information, and has been hypothesized to be partially responsible for the instability of linear attention in long-context tasks (Buckman & Gelada, 2024). Recent works (Sun et al., 2023a; Qin et al., 2023b) obtain better performance through incorporating a global, *non-data-dependent* decay factor[3] $\gamma \in (0,1)$ into linear attention: $\mathbf{S}_t = \gamma \mathbf{S}_{t-1} + \boldsymbol{k}_t^\top \boldsymbol{v}_t$. The use of a single γ is designed to preserve the attention-style parallel form for efficient training. In this work, we consider a data-dependent gating mechanism for linear attention. We show that despite having a more expressive gating factor, the resulting gated linear attention (GLA) layer still admits a hardware-efficient chunkwise form for efficient training.

4.1 Recurrent and Parallel Form of GLA

Recurrent form. GLA has a 2D forget gate $\mathbf{G}_t \in (0,1)^{d_k \times d_v}$ that varies over time:

$$\mathbf{S}_t = \mathbf{G}_t \odot \mathbf{S}_{t-1} + \boldsymbol{k}_t^\top \boldsymbol{v}_t,$$

where we now allow the hidden state to have varying dimensions. This Hadamard product-based recurrent form is very general and encompasses many recent RNNs with 2D hidden states, as listed in Table 1.

Central to the design of gated linear attention is the parameterization of $\mathbf{G}_t$ which requires a balance between *parameter-efficiency*, *state size*, and *training efficiency*. A

[3] This can be viewed as linear attention with ALiBi position encodings (Press et al., 2021). In practice these works also incorporate rotary position embeddings (RoPE; Su et al., 2021).

Model	Parameterization	Learnable parameters
Mamba (Gu & Dao, 2023)	$\mathbf{G}=\exp(-(\mathbf{1}^\top\boldsymbol{\alpha}_t)\odot\exp(\boldsymbol{A}))$, $\boldsymbol{\alpha}_t=\mathrm{softplus}(\boldsymbol{x}_t\boldsymbol{W}_{\alpha_1}\boldsymbol{W}_{\alpha_2})$	$\boldsymbol{A}\in\mathbb{R}^{d_k\times d_v}$, $\boldsymbol{W}_{\alpha_1}\in\mathbb{R}^{d\times\frac{d}{16}}$, $\boldsymbol{W}_{\alpha_2}\in\mathbb{R}^{\frac{d}{16}\times d_v}$
Mamba-2 (Dao & Gu, 2024)	$\mathbf{G}_t=\gamma_t\mathbf{1}^\top\mathbf{1}$, $\gamma_t=\exp(-\mathrm{softplus}(-\boldsymbol{x}_t\boldsymbol{W}_\gamma)\exp(a))$	$\boldsymbol{W}_\gamma\in\mathbb{R}^{d\times 1}$, $a\in\mathbb{R}$
mLSTM (Beck et al., 2024; Peng et al., 2021)	$\mathbf{G}_t=\gamma_t\mathbf{1}^\top\mathbf{1}$, $\gamma_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_\gamma)$	$\boldsymbol{W}_\gamma\in\mathbb{R}^{d\times 1}$
Gated Retention (Sun et al., 2024)	$\mathbf{G}_t=\gamma_t\mathbf{1}^\top\mathbf{1}$, $\gamma_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_\gamma)^{\frac{1}{\tau}}$	$\boldsymbol{W}_\gamma\in\mathbb{R}^{d\times 1}$
DFW (Mao, 2022; Pramanik et al., 2023)	$\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\boldsymbol{\beta}_t$, $\boldsymbol{\alpha}_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_\alpha)$, $\boldsymbol{\beta}_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_\beta)$	$\boldsymbol{W}_\alpha\in\mathbb{R}^{d\times d_k}$, $\boldsymbol{W}_\beta\in\mathbb{R}^{d\times d_v}$
GateLoop (Katsch, 2023)	$\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$, $\boldsymbol{\alpha}_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_{\alpha_1})\exp(\boldsymbol{x}_t\boldsymbol{W}_{\alpha_2}\mathbf{i})$	$\boldsymbol{W}_{\alpha_1}\in\mathbb{R}^{d\times d_k}$, $\boldsymbol{W}_{\alpha_2}\in\mathbb{R}^{d\times d_k}$
HGRN-2 (Qin et al., 2024b)	$\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$, $\boldsymbol{\alpha}_t=\boldsymbol{\gamma}+(1-\boldsymbol{\gamma})\sigma(\boldsymbol{x}_t\boldsymbol{W}_\alpha)$	$\boldsymbol{W}_\alpha\in\mathbb{R}^{d\times d_k}$, $\boldsymbol{\gamma}\in(0,1)^{d_k}$
RWKV-6 (Peng et al., 2024)	$\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$, $\boldsymbol{\alpha}_t=\exp(-\exp(\boldsymbol{x}_t\boldsymbol{W}_\alpha))$	$\boldsymbol{W}_\alpha\in\mathbb{R}^{d\times d_k}$
Gated Linear Attention (GLA)	$\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$, $\boldsymbol{\alpha}_t=\sigma(\boldsymbol{x}_t\boldsymbol{W}_{\alpha_1}\boldsymbol{W}_{\alpha_2})^{\frac{1}{\tau}}$	$\boldsymbol{W}_{\alpha_1}\in\mathbb{R}^{d\times 16}$, $\boldsymbol{W}_{\alpha_2}\in\mathbb{R}^{16\times d_k}$

Table 1: Gated linear attention formulation of recent models, which vary in their parameterization of $\mathbf{G}_t$. The bias terms are omitted.

naïve mapping $\boldsymbol{x}_t\mapsto\mathbf{G}_t$ to obtain a data-dependent gating matrix would require a matrix of size $d\cdot d_k\cdot d_v$, which would be parameter-inefficient. Mao (2022) propose a more efficient outer-product-based low-rank parameterization ($\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\boldsymbol{\beta}_t$), which requires $d\cdot d_v+d\cdot d_k$ parameters.[4]

In Mamba (Gu & Dao, 2023), $\mathbf{G}_t$ is obtained by combining a *data-independent* learnable matrix $\boldsymbol{A}$ with a data-dependent vector $\boldsymbol{\alpha}_t$, which allows the matrix to be full rank. However, this prevents the use of tensor cores because it cannot be reformulated into a matrix-multiply format, as discussed in Dao & Gu (2024). The lack of a compact matrix-multiply form necessitates the materialization of each time step's hidden states. To reduce high I/O costs, Gu & Dao (2023) develop a hardware-aware algorithm that materializes the hidden states exclusively in SRAM rather than in HBM. Due to limited SRAM capacity, this approach cannot scale to larger hidden states, which, as we will show in our experiments, results in suboptimal performance on recall-intensive tasks. Mamba-2 (Dao & Gu, 2024) addresses this limitation with a more restricted gating mechanism: $\mathbf{G}_t=\gamma_t\mathbf{1}^T\mathbf{1}$, where $\gamma_t\in(0,1)$ is a scalar, which makes it possible to to reformulate the recurrence in matrix-multiply form, enabling the use of tensor cores and larger state sizes. This *scalar* data-dependent gating is also used in Peng et al. (2021), Sun et al. (2024), and Beck et al. (2024).

This paper adopts a middle ground between the scalar and the fully low-rank parameterization by using $\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$.[5] This results in the following recurrent form,

$$\mathbf{S}_t=(\boldsymbol{\alpha}_t^\top\mathbf{1})\odot\mathbf{S}_{t-1}+\boldsymbol{k}_t^\top\boldsymbol{v}_t=\mathrm{Diag}(\boldsymbol{\alpha}_t)\mathbf{S}_{t-1}+\boldsymbol{k}_t^\top\boldsymbol{v}_t, \quad (3)$$

where $\boldsymbol{\alpha}_t$ is parameterized via a low-rank linear layer followed by sigmoid on $\boldsymbol{x}_t$ (see §4.4). Note that the above formulation is general and encompasses several recent RNNs (Katsch, 2023; Qin et al., 2024b; Peng et al., 2024). Thus, the hardware-efficient GLA implementation (described next) could be directly used or adapted to other models.

[4]However, Mao (2022) works with only the recurrent form and materializes the hidden states for all time steps in HBM. In Appendix C we give a new algorithm that reformulates the model in a matrix-multiply-based parallel form, which can make use of (an extension of) FLASHLINEARATTENTION for efficient training.

[5]Our preliminary experiments with the $\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\boldsymbol{\beta}_t$ parameterization resulted in only marginal improvements over $\mathbf{G}_t=\boldsymbol{\alpha}_t^\top\mathbf{1}$.

Parallel form. We now describe a parallel form GLA for parallelizing across sequence length. Unrolling Eq. 3 gives

$$\mathbf{S}_t=\sum_{i=1}^{t}\left(\left(\prod_{j=i+1}^{t}\boldsymbol{\alpha}_j^\top\mathbf{1}\right)\odot\boldsymbol{k}_i^\top\boldsymbol{v}_i\right)$$

Letting $\boldsymbol{b}_t:=\prod_{j=1}^{t}\boldsymbol{\alpha}_j$, we can rewrite the above as

$$\begin{aligned}\boldsymbol{o}_t=\boldsymbol{q}_t\mathbf{S}_t&=\boldsymbol{q}_t\sum_{i=1}^{t}\left(\left(\frac{\boldsymbol{b}_t}{\boldsymbol{b}_i}\right)^\top\mathbf{1}\right)\odot\boldsymbol{k}_i^\top\boldsymbol{v}_i\\&=\sum_{i=1}^{t}(\boldsymbol{q}_t\odot\boldsymbol{b}_t)\left(\frac{\boldsymbol{k}_i}{\boldsymbol{b}_i}\right)^\top\boldsymbol{v}_i\end{aligned}$$

where the division is element-wise. Letting $\mathbf{B}\in(0,1)^{L\times d}$ be the matrix obtained from stacking $\boldsymbol{b}_t$'s, the parallel form is:

$$\mathbf{O}=\left(\left(\underbrace{(\mathbf{Q}\odot\mathbf{B})\left(\frac{\mathbf{K}}{\mathbf{B}}\right)^\top}_{\mathbf{P}}\right)\odot\mathbf{M}\right)\mathbf{V}.$$

However, this form is not numerical stable as $\boldsymbol{b}_t$ is the cumulative product of gate values in $\boldsymbol{\alpha}_j\in(0,1)^{1\times d}$, and thus can be extremely small when t is large, making $\frac{\mathbf{K}}{\mathbf{B}}$ explode. To handle this, we can compute in log space for $\mathbf{P}$,[6]

$$\mathbf{P}_{ij}=\sum_{k=1}^{d}\mathbf{Q}_{ik}\mathbf{K}_{jk}\exp(\log\mathbf{B}_{ik}-\log\mathbf{B}_{jk}), \quad i\geq j, \quad (4)$$

where k denotes feature indices. However, unlike vanilla linear attention, as Eq. 4 cannot be represented via a standard matmul, and it cannot make use of half-precision matmuls on tensor cores. We will show in §4.3 how a secondary-level chunking mechanism can enable the use of half-precision matmuls for most computations while maintaining numerical stability, as illustrated in Figure 3.

4.2 Chunkwise Parallel Form of GLA

We derive a chunkwise form of GLA similar to the chunkwise form of basic linear attention (§2.2). Here the intra-chunk operation implements the above parallel form

[6]This form resembles extrapolatable position encoding (Sun et al., 2023b) in that the term inside the exponential can be viewed as a *data-dependent* relative position factor.

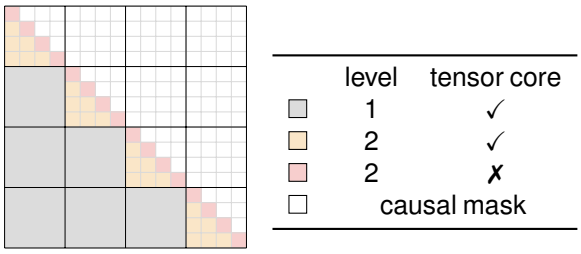

Figure 3: Attention-style map to illustrate the chunkwise computations in GLA. The inter-chunk dependencies (in gray) are not directly computed in the chunkwise form (only computed in the parallel form). The intra-chunk dependencies are modeled via secondary chunking/tiling where the inter-sub-chunk part (in orange) is computed by half-precision matmuls while the intra-sub-chunk part (in pink) is computed in full precision in log space.

at the chunk-level to obtain $\mathbf{O}^{\text{intra}}$. For inter-chunk, we have

$$\boldsymbol{\Lambda}_{iC+j} = \frac{\boldsymbol{b}_{iC+j}}{\boldsymbol{b}_{iC}}, \boldsymbol{\Gamma}_{iC+j} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC+j}}, \boldsymbol{\gamma}_{i+1} = \frac{\boldsymbol{b}_{(i+1)C}}{\boldsymbol{b}_{iC}},$$

$$\mathbf{S}_{[i+1]} = \left(\boldsymbol{\gamma}_{i+1}^{\top}\mathbf{1}\right) \odot \mathbf{S}_{[i]} + \left(\mathbf{K}_{[i+1]} \odot \boldsymbol{\Gamma}_{[i+1]}\right)^{\top} \mathbf{V}_{[i+1]},$$

$$\mathbf{O}_{[i+1]}^{\text{inter}} = \left(\mathbf{Q}_{[i+1]} \odot \boldsymbol{\Lambda}_{[i+1]}\right) \mathbf{S}_{[i]}.$$

Intuitively, $\boldsymbol{\Lambda}_{[i+1]}$ encodes the cumulative decay from the start of a chunk which will be used to propagate the hidden states from the previous chunk $\mathbf{S}_{[i]}$, while $\boldsymbol{\Gamma}_{[i+1]}$ encodes the decay to the end of a chunk which will be used to accumulate information to be added to the next hidden state $\mathbf{S}_{[i+1]}$.

4.3 Hardware-Efficient GLA

With the chunkwise form in hand, we can adapt the FLASHLINEAR ATTENTION algorithm presented in §3 to the gated case. The adaptation additionally relies on two crucial techniques described below. We give high-level intuitions in this section and defer the full algorithms to Alg. 3-6 of Appendix A.3.

Secondary-level chunking. Unlike in ordinary linear attention, the intra-chunk computations in GLA cannot leverage half-precision matmuls (and thus tensor cores) due to log space computations (Eq. 4). To make better use of tensor cores, we use secondary-level chunking scheme, where a chunk is further divided into sub-chunks (i.e., another level of tiling) in the spirit of classic tiling techniques (Dao et al., 2022b). The attention-like matrix $\mathbf{P} \in \mathbb{R}^{L \times L}$ is then computed in a chunkwise manner, as illustrated in Figure 3. Concretely, the interactions between sub-chunks are computed via half-precision matmuls,[7]

$$\mathbf{P}_{[i][j]} = \left(\mathbf{Q}_{[i]} \odot \boldsymbol{\Lambda}_{[i]}\right)\left(\mathbf{K}_{[j]} \odot \boldsymbol{\Gamma}_{[j]} \odot \frac{\boldsymbol{b}_{iC}}{\boldsymbol{b}_{(j+1)C}}\right)^{\top} \in \mathbb{R}^{C \times C}.$$

This corresponds to the orange tiles in Figure 3. For the intra-sub-chunk part (pink tiles in Figure 3) we have to resort to Eq. 4 and perform the matmul in full precision for stability. With this two-level tiling strategy, the total amount of non-half-precision matmul FLOPs are greatly reduced, thus leading to wallclock improvements. We provide the Pytorch-style pseudo-code in Listing 1 of Appendix A.3.

Memory-efficient $\mathbf{d}\boldsymbol{\alpha}_t$ computation. Past work (Mao, 2022, §3.1) has claimed that GLA-like models have to materialize the matrix-valued hidden states of size $L \times d \times d$ in HBM to compute all the gradients $\mathbf{d}\boldsymbol{\alpha}_t$, since $\mathbf{d}\boldsymbol{\alpha}_t = (\mathbf{S}_{t-1} \odot \mathbf{d}\mathbf{S}_t)\mathbf{1}$. We instead give the following *closed form* formula for $\mathbf{d}\log\boldsymbol{\alpha}_t$,

$$\mathbf{d}\log\boldsymbol{b}_t = \boldsymbol{q}_t \odot \mathbf{d}\boldsymbol{q}_t - \boldsymbol{k}_t \odot \mathbf{d}\boldsymbol{k}_t, \quad \mathbf{d}\log\boldsymbol{\alpha}_t = \sum_{t \le i \le L} \mathbf{d}\log\boldsymbol{b}_i,$$

which can be easily obtained by taking the derivative with respect to Eq. 4 (see Appendix A.3 for full derivation). $\mathbf{d}\boldsymbol{q}_t$ and $\mathbf{d}\boldsymbol{k}_t$ can be computed as in Alg. 2.

4.4 GLA Transformer

We generalize the GLA layer to the multi-head case. Given H heads, we have the following for each head $h \in [1, H]$,

$$\mathbf{S}_t^h = \left(\left(\boldsymbol{\alpha}_t^h\right)^{\top}\mathbf{1}\right) \odot \mathbf{S}_{t-1}^h + \boldsymbol{k}_t^{h\top}\boldsymbol{v}_t^h \in \mathbb{R}^{d_k' \times d_v'},$$

$$\boldsymbol{o}_t^h = \boldsymbol{q}_t^h \mathbf{S}_t^h \in \mathbb{R}^{1 \times d_v'},$$

$$\boldsymbol{o}_t' = \text{concat}(\text{LN}(\boldsymbol{o}_t^1), \ldots, \text{LN}(\boldsymbol{o}_t^H)) \in \mathbb{R}^{1 \times d_v},$$

$$\boldsymbol{r}_t = \text{Swish}(\boldsymbol{x}_t \boldsymbol{W}_r + \boldsymbol{b}_r) \in \mathbb{R}^{1 \times d_v},$$

$$\boldsymbol{y}_t = (\boldsymbol{r}_t \odot \boldsymbol{o}_t')\boldsymbol{W}_O \in \mathbb{R}^{1 \times d}.$$

Here we use separate key (d_k) and value (d_v) dimensions; $d_k' = d_k/H, d_v' = d_v/H$ are the per-head key/value dimensions. LayerNorm (LN) is applied after the output of each head, while the output projection and output gating operate on the concatenation of head outputs (Sun et al., 2023a).

We then build up a Transformer-like model by interleaving multi-head GLA layers with feed-forward networks (FFN). Concretely, given layer l's contextualized representation $\mathbf{X}^{(l)}$, we obtain $\mathbf{X}^{(l+1)}$ via,

$$\mathbf{Y}^{(l)} = \text{GLA}(\text{LN}(\mathbf{X}^{(l)})) + \mathbf{X}^{(l)}$$

$$\mathbf{X}^{(l+1)} = \text{SwiGLU}(\text{LN}(\mathbf{Y}^{(l)})) + \mathbf{X}^{(l)},$$

where the SwiGLU FFN layer (Touvron et al., 2023) is,

$$\text{SwiGLU}(\mathbf{Z}) = (\text{Swish}(\mathbf{Z}\boldsymbol{W}_1) \odot \mathbf{Z}\boldsymbol{W}_2)\boldsymbol{W}_3.$$

Parameter allocation. As presented, our GLA layer employs two additional matrices for predicting $\boldsymbol{\alpha}_t, \boldsymbol{r}_t$ (i.e., $\boldsymbol{W}_\alpha, \boldsymbol{W}_r$) compared to a regular softmax attention layer. For parameter-efficiency, we use a low-rank parameterization

$$\boldsymbol{\alpha}_t = \sigma((\boldsymbol{x}_t \boldsymbol{W}_\alpha^1 \boldsymbol{W}_\alpha^2 + \boldsymbol{b}_\alpha))^{\frac{1}{\tau}} \in \mathbb{R}^{1 \times d_k},$$

where $\boldsymbol{W}_\alpha^1 \in \mathbb{R}^{d \times 16}$, $\boldsymbol{W}_\alpha^2 \in \mathbb{R}^{16 \times d_k}$, and $\tau = 16$ is a temperature term to encourage model to have a slower forgetting rate. We further set $d_k = \frac{d}{2}$ and $d_v = d$ and use full-rank parameterizations for $(\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V, \boldsymbol{W}_O, \boldsymbol{W}_r)$. Ultimately, one GLA layer collectively needs (roughly) $4d^2$ parameters, as in regular softmax attention.

[7] To reduce notational clutter, here we use the notations from the first-level chunking to express the key idea. The actual implementation is done with secondary-level chunks.

Scale	Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_norm ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_norm ↑	Avg. ↑
340M Params	Transformer++	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	41.2
15B Tokens	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
	GLA	28.65	43.35	30.3	64.8	34.5	51.4	45.1	22.7	41.5
1.3B Params	Transformer++	16.85	13.44	48.9	70.8	49.6	53.6	56.0	26.5	50.9
100B Tokens	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	17.22	14.47	46.9	71.8	49.8	53.9	57.2	26.6	51.0

Table 2: GLA Transformer results against Transformer++ (Touvron et al., 2023), RetNet (Sun et al., 2023a), and Mamba (Gu & Dao, 2023). All models are trained on the same subset of the SlimPajama dataset with the Mistral tokenizer. The 340M/1.3B models are trained for 15B/100B tokens respectively. The individual task performance is via zero-shot. We report the main results on the same set of tasks reported by Gu & Dao (2023). See Appendix D for results on other benchmarks, including 5-shot results. The last column shows the average over all benchmarks that use (normalized) accuracy as the metric.

5 Empirical Study

5.1 Experimental Setup

Our main experiments are on language modeling, where we study whether GLA can perform competitively against a (i) strong Transformer baseline with modern architectural recipes and (ii) recent linear-time models. We use the SlimPajama dataset (Soboleva et al., 2023) and tokenize it using the Mistral tokenizer (Jiang et al., 2023). The original dataset contains 627B tokens; we use a 100B subset.

Baselines. We evaluate GLA against three baselines: Transformer++ (Touvron et al., 2023), RetNet (Sun et al., 2023a), and Mamba (Gu & Dao, 2023). Transformer++ is the LLaMA architecture with Rotary Positional Embeddings (Su et al., 2021), SWiGLU (Shazeer, 2020), and RMSNorm (Zhang & Sennrich, 2019); we also use SwiGLU in the RetNet to replace its original FFN for fair comparison. For Mamba, we use the open source code. All our baselines are trained for the exact same number of tokens on the same dataset for fair comparison.

Training details. We train all models from scratch at two scales: 340M and 1.3B. All models are trained with AdamW (Loshchilov & Hutter, 2018) using a maximum learning rate of 3e-4. The 340M models are trained on 15B tokens with a batch size of 0.5M tokens, while the 1.3B models are trained on 100B tokens with a batch size of 2M tokens. We use a cosine learning rate schedule with a warmup of 0.5B/1B tokens for the 340M/1.3B settings, respectively. The initial and final learning rates are 3e-5. We use a weight decay of 0.01, and gradient clipping of 1.0.

5.2 Main Results

In addition to perplexity (ppl) on Wikitext (Wiki.), we consider a wide range of downstream tasks covering common-sense reasoning and question-answering as was used in Gu & Dao (2023): LAMBADA (LMB.; Paperno et al., 2016), PiQA (Bisk et al., 2020), HellaSwag (Hella.; Zellers et al., 2019), WinoGrande (Wino.; Sakaguchi et al., 2021), ARC-easy (ARC-e) and ARC-challenge (Arc-c) (Clark et al., 2018). In Appendix D, we also include results on additional tasks: Copa (Roemmele et al., 2011), SciQA (Auer et al., 2023), OpenbookQA (Mihaylov et al., 2018), BoolQA (Clark et al., 2019). We report perplexity (ppl) on WikiText and LAMBADA, accuracy normalized by length on HellaSwag, ARC-challenge and OpenbookQA, and accuracy on the other tasks. All evaluations are performed using the LM evaluation harness (Gao et al., 2021).

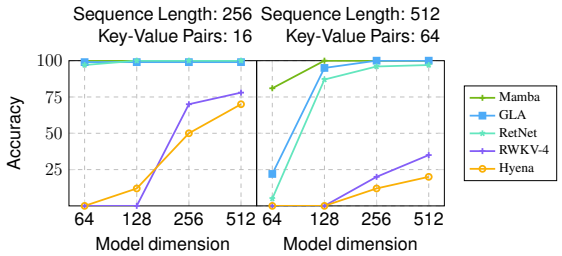

Figure 4: Accuracy (%) on the synthetic MQAR task.

Our main results are shown in Table 2. Compared to RetNet which uses a data-independent decay rate, the GLA Transformer with data-dependent gates shows improved results on all tasks. Both GLA Transformer and Mamba show comparable performance to Transformer++.

Recall-intensive tasks. While subquadratic models can achieve competitive language modeling performance to Transformers, Arora et al. (2024) show that they lag behind softmax attention in recall-intensive tasks. We next evaluate GLA on real and synthetic tasks that focus on recall.

The synthetic MQAR task (Arora et al., 2023a) is a more challenging multi-query version of the induction head task (Fu et al., 2023b) in which a model has to recall the token following a query token multiple times. We follow Arora et al. (2023a)'s experimental setting and compare GLA against recent subquadractic models, including RetNet (Sun et al., 2023a), Mamba (Gu & Dao, 2023), Hyena (Poli et al., 2023) and RWKV-4 (Peng et al., 2023). For RetNet and GLA the number of heads is set to 2; for other models we follow the default settings in Arora et al. (2023a). The results are shown

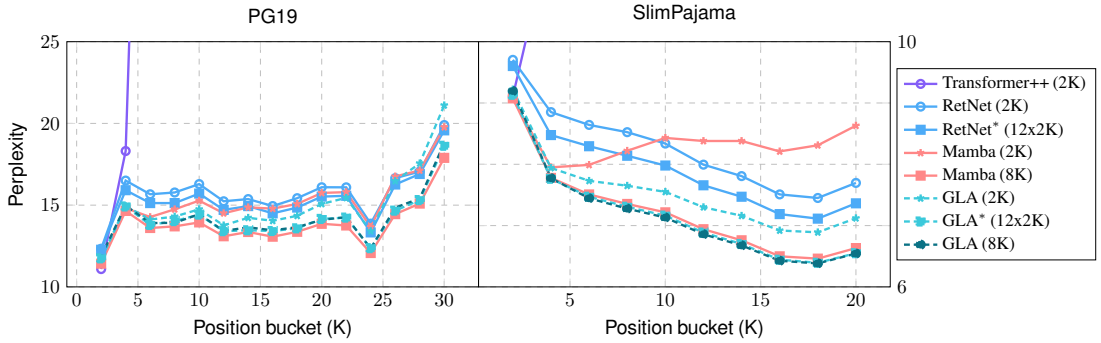

Figure 5: Length extrapolation on the test set of SlimPajama and PG19. We pretrain 1.3B models from scratch on SlimPajama for 100B tokens with different training length. * indicates models using truncated BPTT with over 12 segments that are each of 2K-length.

Scale	Model	FDA	SWDE	SQUAD
340M Params	Transformer++	21.4	42.2	22.1
15B Tokens	RetNet	2.9	13.3	27.6
	Mamba	2.1	12.4	23.0
	GLA	8.1	18.6	27.2
1.3B Params	Transformer++	27.4	66.6	31.5
100B Tokens	RetNet	14.3	42.8	34.7
	Mamba	6.2	41.4	35.2
	GLA	19.9	50.6	42.6

Table 3: Comparison of different models in three recall-intensive tasks tested in Arora et al. (2024). Higher is better for all tasks.

in Figure 4. Standard quadratic attention achieves perfect scores in all settings and is thus omitted. We find that models with matrix-valued hidden states (i.e., Mamba/RetNet/GLA) outperform Hyena/RWKV, and our GLA outperforms RetNet, confirming the benefits of using data-dependent gates.

Following Arora et al. (2024), we also test our models on three real recall-intensive tasks: FDA (Arora et al., 2023b), SWDE (Lockard et al., 2019), and SQUAD (Rajpurkar et al., 2018). These tasks focus on information extraction or reading comprehension. As illustrated in Table 3, subquadratic models significantly underperform Transformers on the FDA and SWDE, both of which are information extraction tasks. However, GLA outperforms other subquadractic models, likely due to its larger recurrent state (compared to Mamba) and selection mechanism (compared to RetNet).

Long sequence training and length extrapolation. One advantage of linear attention models is that they allow for efficient long sequence training in linear time. To showcase this feature, we consider two training settings: (i) direct training on 8K-length contexts, (ii) training on 24K-length contexts through truncated backpropagation through time (TBPP) over 2K-length segments.[8] In the latter case the gradients are not back-propagated across segments, and hence this approach has minimal overhead comparable to the standard 2K-length training strategy (where the initial hidden state is always set to zero). We pretrain 1.3B Mamba, RetNet, and GLA models on SlimPajama for 100B tokens on these settings and test them on both SlimPajama test set and PG19 (Rae et al., 2019) test set.

Figure 5 shows the perplexities of the tokens calculated in different position groups. For models trained on 2K-length contexts, GLA extrapolates better than Mamba/RetNet in most position buckets on the PG19 test set; Mamba struggles to extrapolate beyond 4K, while GLA/RetNet can generalize to 18K on the Slimpajama test set. Transformers cannot extrapolate beyond training length, which is a known failure mode.[9] Pretraining in a long sequence consistently improves perplexities for all three models. We found marginal perplexity difference in the two settings for GLA, indicating that TBPTT might be a more economic approach to long-sequence training. Mamba benefits significantly from 8K-length training, and it performs similarly as GLA in the same training setting.

Ablations. We conduct a small-scale ablation study by training the 340M GLA variants for 7B tokens. We investigate (i) the importance of having both *fine-grained* and *data-dependent* gating and (ii) the influence of head dimension size. The results are shown in Table 4. For (i), we find that while data dependent scalar gates substantially improve upon RetNet, a finer-grained gating mechanism is still necessary. For (ii) we tune the number of heads to vary head dimensions, where by default GLA uses 4 heads. Increasing it to 8 (i.e., smaller head dimension) leads to relatively large perplexity degradation; reducing it to 1 (i.e., larger head dimension) actually performs best, but results in only marginal improvement while requiring much higher GPU memory. We thus choose 4 heads for our experiments.

5.3 Training Efficiency

Fig. 6 shows the throughput and memory usage as a function of the sequence length and batch size for the different 1.3B models on a single H100 GPU.[10] Here GLA adopts the

[8] We split a 24K input sequence into 12 segments. The final state of the previous segment is used as the initial state for the current segment.

[9] Although there are positional encoding schemes that enable better length extrapolation, these methods still have difficulty generalizing significantly beyond context lengths seen during training (Press et al., 2021; Sun et al., 2023b; Li et al., 2023c).

[10] We use the official implementation for Mamba, the fused version of SwiGLU for Transformer++ and GLA, and FlashAttention-2

Model variants	Training ppl.
GLA Transformer (4 heads)	14.77
No gate (i.e., Linear Attention)	23.21
Data independent scalar decay (i.e., RetNet)	16.55
Data dependent scalar gate	15.56
Small head dimension (8 heads)	15.29
Large head dimension (1 head)	14.61

Table 4: Ablation study results on the 340M model trained for 7B tokens. We evaluate the model variants via the average perplexity of the last 200 training steps.

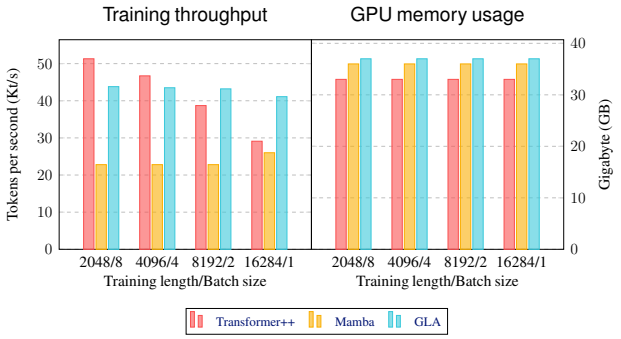

Figure 6: Training throughput and memory footprint on an H100.

materialization version of FLASHLINEARATTENTION with recomputation of hidden state (§3.3). All models have linear space complexity, and the total GPU footprint difference among them is minimal. In terms of training throughput, Mamba lags behind Transformer++ and GLA, with GLA shows greater advantages in training lengths beyond 4096.

5.4 Limitations & Future Work

While our experiments with the GLA Transformer were on a respectable scale, we were unable to perform larger-scale experiments due to limited compute resources. Although it is unclear at this point how GLA would scale to even larger models/datasets, we anticipate that training efficiency of GLA become even more favorable compared to Mamba at larger scales. Specifically, when scaled to larger sizes (e.g., $>$ 7B), GLA can be more efficient than Mamba because of better use of tensor cores and GLA's compatibility with tensor parallelism.[11] Insofar as we are interested in leveraging the efficiency of linear attention, it would be interesting to apply GLA to other modalities (especially modalities with long-range dependencies), in line with recent work on applying state-of-the-art state-space models to other types of data (Yan et al., 2023; Zhu et al., 2024; Ma et al., 2024; Liu et al., 2024; Xing et al., 2024; Wang et al., 2024a;b; Yang et al., 2024, *inter alia*).

6 Related Work

We briefly discuss related work here and give an extended discussion of the related work in Appendix A.

Traditional RNNs are difficult to scale due to the nonlinear dependencies between the hidden states and expensive matmul-based sequential hidden state updates. Linear RNNs/State-Space Models (SSMs)/Transformers eliminate nonlinear dependencies, making training parallelizable along the temporal dimension (Martin & Cundy, 2018; Gu et al., 2022; Smith et al., 2023). Such models have been the focus of much recent work as a competitive sub-quadratic alternative to the Transformer architecture (Peng et al., 2023; Gu & Dao, 2023; Qin et al., 2023c;b; Sun et al., 2023a; Wang et al., 2022).

Data-dependent decay rates have always been regarded important for RNNs (Gers et al., 2000; van der Westhuizen & Lasenby, 2018). Typical forget gate values depend on both the previous hidden state and the current input. However Martin & Cundy (2018) suggest that forget gate values should depend solely on the current inputs to enable parallel training. This simple strategy has been shown to be effective in moderate-scale experiments conducted by HGRN (Qin et al., 2023b). RWKV-v6 (Peng et al., 2024) and Mamba (Gu & Dao, 2023) also use data-dependent decay rates that are reminiscent of forget gates. In the context of linear Transformers, Peng et al. (2021) employ a coarse-grained position-wise forget gate, while Mao (2022) and Katsch (2023) use a more fine-grained forget gate.

RNNs rely on fixed-dimensional hidden states to encode their entire history. The hidden state dimension serves as a proxy for memory capacity and thus significantly influences their expressive power. Linear Transformers expand the hidden dimension of RNNs via the outer-product parameterization, as discussed §2.1. Linear SSMs on the other hand expand their hidden dimension via a single-input-single-output (SISO) strategy. Without data-dependent SSM parameters, this can be done efficiently during training via the Fast Fourier Transform (FFT). However, with data-dependent SSM parameters, FFT-based training is not possible, and thus Gu & Dao (2023) implements a custom CUDA kernel to train a selective state-space model using the parallel scan algorithm (Smith et al., 2023). To fit all the hidden states into SRAM, they can only afford an expansion rate up to 16. In contrast our hardware-aware training algorithm provides an alternative, efficient approach for expanding the hidden dimension to a wider range, which we have shown useful in recall-intensive tasks.

7 Conclusion

We propose an efficient algorithm for training linear attention Transformers with data-dependent gating mechanisms. Our algorithm makes it possible to balance FLOPs against parallellism, while still allowing for the use of half-precision matmuls which can take advantage of tensor core units on modern GPUs. Experiments on language modeling demonstrate that gated linear attention Transformers can perform respectably compared to strong baselines.

for Transformer++.

[11] In particular, since Mamba is not a multi-head model it is not as amenable to tensor parallelism.

Impact Statement

This paper aims to improve the training efficiency of a new model family of (gated) linear attention models. The efficiency advantage of such models might help democratize access of language models. On the other hand, whether such new architectures would affect known issues such as biased and harmful outputs of language models remains an unexplored research question.

Acknowledgments

This work was supported by MIT-IBM Watson AI. We thank Yutao Sun, Zhen Qin, Li Dong, Xinyu Yang, Jiacheng You, Huanqi Cao, Yu Zhang, and Shida Wang for their insightful discussions. We also thank Yu Zhang, Fares Obeid, Daniel Goldstein, and Liliang Ren for their proofreading. Special thanks to Yu Zhang for contributing to the FLASHLINEARATTENTION library.

References

Arora, S., Eyuboglu, S., Timalsina, A., Johnson, I., Poli, M., Zou, J., Rudra, A., and Ré, C. Zoology: Measuring and improving recall in efficient language models. *CoRR*, abs/2312.04927, 2023a.

Arora, S., Yang, B., Eyuboglu, S., Narayan, A., Hojel, A., Trummer, I., and Ré, C. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes, April 2023b. URL http://arxiv.org/abs/2304.09433. arXiv:2304.09433 [cs].

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and R'e, C. Simple linear attention language models balance the recall-throughput tradeoff. *ArXiv*, abs/2402.18668, 2024.

Auer, S., Barone, D. A. C., Bartz, C., Cortes, E. G., Jaradeh, M. Y., Karras, O., Koubarakis, M., Mouromtsev, D., Pliukhin, D., Radyush, D., Shilin, I., Stocker, M., and Tsalapati, E. The sciqa scientific question answering benchmark for scholarly knowledge. *Scientific Reports*, 13(1):7240, May 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-33607-z.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv: Arxiv-2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150v2.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Blelloch, G. E. Prefix sums and their applications. 1990.

Brandon, W., Nrusimha, A., Qian, K., Ankner, Z., Jin, T., Song, Z., and Ragan-Kelley, J. Striped attention: Faster ring attention for causal transformers. *ArXiv*, abs/2311.09431, 2023.

Buckman, J. and Gelada, C. Linear Transformers Are Faster After All, 2024.

Chaurasia, G., Ragan-Kelley, J., Paris, S., Drettakis, G., and Durand, F. Compiling high performance recursive filters. In *High Performance Graphics*, 2015.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *PREPRINT*, 2019. URL https://arxiv.org/abs/1904.10509v1.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *CoRR*, abs/2307.08691, 2023. doi: 10.48550/ARXIV.2307.08691.

Dao, T. and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024.

Dao, T., Chen, B., Sohoni, N. S., Desai, A. D., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, 2022a.

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

Fu, D. Y., Arora, S., Grogan, J., Johnson, I., Eyuboglu, S., Thomas, A. W., Spector, B., Poli, M., Rudra, A., and R'e, C. Monarch mixer: A simple sub-quadratic gemm-based architecture. *ArXiv*, abs/2310.12109, 2023a.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b.

Fu, D. Y., Epstein, E. L., Nguyen, E., Thomas, A., Zhang, M., Dao, T., Rudra, A., and Ré, C. Simple hardware-efficient long convolutions for sequence modeling. *International Conference on Machine Learning*, 2023c. doi: 10.48550/arXiv.2302.06646. URL https://arxiv.org/abs/2302.06646v1.

Fu, D. Y., Kumbong, H., Nguyen, E., and Ré, C. Flashfftconv: Efficient convolutions for long sequences with tensor cores. *CoRR*, abs/2311.05908, 2023d.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021.

Gers, F. A., Schmidhuber, J., and Cummins, F. A. Learning to forget: Continual prediction with LSTM. *Neural Comput.*, 12(10):2451–2471, 2000.

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

Gu, A., Goel, K., and R'e, C. Efficiently modeling long sequences with structured state spaces. *International Conference On Learning Representations*, 2021a.

Gu, A., Johnson, I., Goel, K., Saab, K. K., Dao, T., Rudra, A., and R'e, C. Combining recurrent, convolutional, and continuous-time models with linear state-space layers. *Neural Information Processing Systems*, 2021b. URL https://arxiv.org/abs/2110.13985v1.

Gu, A., Goel, K., and Ré, C. 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, 2022.

Gupta, A. and Berant, J. Diagonal state spaces are as effective as structured state spaces. *ARXIV.ORG*, 2022. doi: 10.48550/arXiv.2203.14343.

Hasani, R., Lechner, M., Wang, T.-H., Chahine, M., Amini, A., and Rus, D. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

Hinton, G. E. and Plaut, D. C. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pp. 177–186, 1987.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Hooker, S. The hardware lottery. *Communications of the ACM*, 64:58 – 65, 2020.

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

Irie, K., Schlag, I., Csordás, R., and Schmidhuber, J. Going beyond linear transformers with recurrent fast weight programmers. *Advances in Neural Information Processing Systems*, 34:7703–7717, 2021.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *ArXiv preprint*, abs/2310.06825, 2023.

Kacham, P., Mirrokni, V., and Zhong, P. Polysketchformer: Fast transformers via sketching polynomial kernels, 2023.

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. Finetuning pretrained transformers into RNNs. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10630–10643, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.830.

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

Katsch, T. Gateloop: Fully data-controlled linear recurrence for sequence modeling. *ArXiv*, abs/2311.01927, 2023.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. *International Conference On Learning Representations*, 2020. URL https://arxiv.org/abs/2001.04451v2.

Li, D., Shao, R., Xie, A., Xing, E. P., Gonzalez, J. E., Stoica, I., Ma, X., and Zhang, H. Lightseq: Sequence level parallelism for distributed training of long context transformers. *ArXiv*, abs/2310.03294, 2023a.

Li, S., Xue, F., Baranwal, C., Li, Y., and You, Y. Sequence parallelism: Long sequence training from system perspective. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada, July 2023b. Association for Computational Linguistics.

Li, S., You, C., Guruganesh, G., Ainslie, J., Ontanon, S., Zaheer, M., Sanghai, S., Yang, Y., Kumar, S., and Bhojanapalli, S. Functional interpolation for relative positions improves long context transformers. *arXiv preprint arXiv:2310.04418*, 2023c.

Li, Y., Cai, T., Zhang, Y., Chen, D., and Dey, D. What makes convolutional models great on long sequence modeling? In *The Eleventh International Conference on Learning Representations*, 2023d. URL https://openreview.net/forum?id=TGJSPbRpJX-.

Lingle, L. D. Transformer-vq: Linear-time transformers via vector quantization. *CoRR*, abs/2309.16354, 2023. doi: 10.48550/ARXIV.2309.16354.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context. *ArXiv*, abs/2310.01889, 2023.

Liu, Y., Tian, Y., Zhao, Y., Yu, H., Xie, L., Wang, Y., Ye, Q., and Liu, Y. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024.

Lockard, C., Shiralkar, P., and Dong, X. L. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3047–3056, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1309. URL https://aclanthology.org/N19-1309.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. 2018.

Ma, J., Li, F., and Wang, B. U-mamba: Enhancing long-range dependency for biomedical image segmentation. *arXiv preprint arXiv:2401.04722*, 2024.

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., and Zettlemoyer, L. Mega: Moving average equipped gated attention. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=qNLe3iq2El.

Mao, H. H. Fine-tuning pre-trained transformers into decaying fast weights. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 10236–10242, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.697.

Martin, E. and Cundy, C. 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, 2018.

Massaroli, S., Poli, M., Fu, D. Y., Kumbong, H., Parnichkun, R. N., Timalsina, A., Romero, D. W., McIntyre, Q., Chen, B., Rudra, A., Zhang, C., Re, C., Ermon, S., and Bengio, Y. Laughing hyena distillery: Extracting compact recurrences from convolutions. *NEURIPS*, 2023. URL https://arxiv.org/abs/2310.18780v1.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Nahshan, Y., Kampeas, J., and Haleva, E. Linear log-normal attention with unbiased concentration, 2023.

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *ArXiv*, abs/2401.06104, 2024.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

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. RWKV: reinventing rnns for the transformer era. *CoRR*, abs/2305.13048, 2023. doi: 10.48550/ARXIV.2305.13048.

Peng, B., Goldstein, D., Anthony, Q., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Ferdinan, T., Hou, H., Kazienko, P., et al. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 2024.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

Peng, H., Kasai, J., Pappas, N., Yogatama, D., Wu, Z., Kong, L., Schwartz, R., and Smith, N. A. ABC: Attention with bounded-memory control. In Muresan, S., Nakov, P., and Villavicencio, A. (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland, May 2022. Association for Computational Linguistics.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 28043–28078. PMLR, 2023.

Pramanik, S., Elelimy, E., Machado, M. C., and White, A. Recurrent linear transformers. *CoRR*, abs/2310.15719, 2023.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

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

Qin, Z., Han, X., Sun, W., He, B., Li, D., Li, D., Dai, Y., Kong, L., and Zhong, Y. Toeplitz neural network for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023a. URL https://openreview.net/forum?id=IxmWsm4xrua.

Qin, Z., Li, D., Sun, W., Sun, W., Shen, X., Han, X., Wei, Y., Lv, B., Yuan, F., Luo, X., et al. Scaling transnormer to 175 billion parameters. *arXiv preprint arXiv:2307.14995*, 2023b.

Qin, Z., Yang, S., and Zhong, Y. Hierarchically gated recurrent neural network for sequence modeling. *CoRR*, abs/2311.04823, 2023c. doi: 10.48550/ARXIV.2311.04823.

Qin, Z., Sun, W., Li, D., Shen, X., Sun, W., and Zhong, Y. Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. 2024a.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024b.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019.

Rajpurkar, P., Jia, R., and Liang, P. Know What You Don't Know: Unanswerable Questions for SQuAD. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

Ren, L., Liu, Y., Wang, S., Xu, Y., Zhu, C., and Zhai, C. Sparse modular activation for efficient sequence modeling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=TfbzX6I14i.

Roemmele, M., Bejan, C. A., and Gordon, A. S. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011. URL https://people.ict.usc.edu/~gordon/publications/AAAI-SPRING11A.PDF.

Romero, D. W., Kuzina, A., Bekkers, E. J., Tomczak, J. M., and Hoogendoorn, M. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv: 2102.02611*, 2021. URL https://arxiv.org/abs/2102.02611v3.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *International Conference On Topology, Algebra And Categories In Logic*, 2020. doi: 10.1162/tacl_a_00353. URL https://arxiv.org/abs/2003.05997v5.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Saphra, N., Fleisig, E., Cho, K., and Lopez, A. First tragedy, then parse: History repeats itself in the new era of large language models. *ArXiv*, abs/2311.05020, 2023.

Schlag, I., Irie, K., and Schmidhuber, J. Linear transformers are secretly fast weight programmers. In Meila, M. and Zhang, T. (eds.), *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*, pp. 9355–9366. PMLR, 2021.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Smith, J. T. H., Warrington, A., and Linderman, S. W. 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, 2023.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021.

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

Sun, Y., Dong, L., Patra, B., Ma, S., Huang, S., Benhaim, A., Chaudhary, V., Song, X., and Wei, F. A length-extrapolatable transformer. In Rogers, A., Boyd-Graber, J. L., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 14590–14604. Association for Computational Linguistics, 2023b. doi: 10.18653/V1/2023.ACL-LONG.816.

Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., and Wei, F. You only cache once: Decoder-decoder architectures for language models. 2024.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

van der Westhuizen, J. and Lasenby, J. The unreasonable effectiveness of the forget gate. *CoRR*, abs/1804.04849, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, C., Tsepa, O., Ma, J., and Wang, B. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces. *arXiv preprint arXiv:2402.00789*, 2024a.

Wang, J., Yan, J. N., Gu, A., and Rush, A. M. Pretraining without attention. *CoRR*, abs/2212.10544, 2022.

Wang, J., Gangavarapu, T., Yan, J. N., and Rush, A. M. Mambabyte: Token-free selective state space model. *arXiv preprint arXiv:2401.13660*, 2024b.

Wu, F., Fan, A., Baevski, A., Dauphin, Y., and Auli, M. Pay less attention with lightweight and dynamic convolutions. *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1901.10430v2.

Xing, Z., Ye, T., Yang, Y., Liu, G., and Zhu, L. Segmamba: Long-range sequential modeling mamba for 3d medical image segmentation. *arXiv preprint arXiv:2401.13560*, 2024.

Yan, J. N., Gu, J., and Rush, A. M. Diffusion models without attention. 2023.

Yang, Y., Xing, Z., and Zhu, L. Vivim: a video vision mamba for medical video object segmentation. *arXiv preprint arXiv:2401.14168*, 2024.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020. URL https://arxiv.org/abs/2007.14062v2.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, J., Jiang, S., Feng, J., Zheng, L., and Kong, L. Linear attention via orthogonal memory, 2023.

Zhang, M., Bhatia, K., Kumbong, H., and Ré, C. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry, 2024.

Zhang, Y. and Cai, D. Linearizing transformer with key-value memory. In Goldberg, Y., Kozareva, Z., and Zhang, Y. (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

Zhu, L., Liao, B., Zhang, Q., Wang, X., Liu, W., and Wang, X. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024.

A Extended Related Work

A.1 Linear Attention

Feature map ϕ. Linear attention mechanisms (Katharopoulos et al., 2020) replace $\exp(\boldsymbol{q}_t\boldsymbol{k}_i^\top)$ with a kernel $k(\boldsymbol{x},\boldsymbol{y})$ having an associated feature map ϕ (i.e., $k(\boldsymbol{x},\boldsymbol{y}) = \langle\phi(\boldsymbol{x}),\phi(\boldsymbol{y})\rangle$) where $\phi \in \mathbb{R}^{d_{\text{key}}} \to \mathbb{R}^{d_{\text{dot}}}$. ϕ often consists of two parts: $\phi = \phi_0 \circ \phi_1$. ϕ_1 could be a linear map made up by random samples (Peng et al., 2021; Choromanski et al., 2021), learnable MLPs (Kasai et al., 2021; Zhang et al., 2024; Kacham et al., 2023) or simply an identity map (Mao, 2022). ϕ_2 is often an element-wise (activation) function that makes the resulting ϕ a positive feature map, such as $1+\mathrm{elu}$ (Katharopoulos et al., 2020), ReLU (Kasai et al., 2021), $\exp(\cdot)$ (Zhang et al., 2024; Choromanski et al., 2021). Some work (Qin et al., 2023b; Sun et al., 2023a; Mao, 2022) suggests that a positive feature map might not be necessary.

Our work follows Sun et al. (2023a) and Mao (2022) by using an identity map $\phi = \mathbf{I}$. Recent work suggests that non-identity feature maps such as scaled element-wise exponential map (Nahshan et al., 2023; Zhang et al., 2024) and higher-order polynomial map (Arora et al., 2024; Kacham et al., 2023) work well empirically. We leave the exploration of integrating other types of feature map into GLA to future work.

Attention spikiness. Linear attention suffers from the "attention dilution" issue (Qin et al., 2022), where the attention distribution is too uniform (i.e., high entropy) to concentrate on relevant tokens. Qin et al. (2022) propose adding local attention layers to focus more on adjacent tokens, a method adopted in (Lingle, 2023; Nahshan et al., 2023; Zhang et al., 2023) and proven crucial for performance. Recent work finds that a scaled element-wise exponential map—i.e., $\phi(\mathbf{x}) = \exp(t \cdot \mathbf{x})$ with $t \geq 2$—helps to concentrate attention (Nahshan et al., 2023; Zhang et al., 2024). Zhang et al. (2024) also find that higher-order polynomial kernels induce low-entropy and spiky attention distribution, partially explaining the empirical success of Based Linear Attention (Arora et al., 2024) and PolySketchFormer (Kacham et al., 2023).

Memory capacity. Linear attention has bounded memory size (Peng et al., 2022) while softmax attention enjoys unbounded memory(Oren et al., 2024). We believe that increasing the memory size efficiently and utilizing memory effectively are the keys to bridging the performance gap between linear attention and softmax attention. To increase memory size, it is shown that directly increasing d_{key} is effective (Sun et al., 2023a; Mao, 2022; Zhang & Cai, 2022); however, the total parameters are hard to control with the increase of d_{key}. Parameter-efficient methods often keep d_{key} intact and increase d_{dot} instead. Higher order polynomial kernels with order $p \geq 2$ map d_{key} to a much higher $d_{\text{dot}} = O(d_{\text{key}}^p)$ (Arora et al., 2023a; Kacham et al., 2023). Schlag et al. (2021) propose the Deterministic Parameter-Free Projection (DPFP), while Pramanik et al. (2023) use parameterized outer product to expand d_{dot} in a parameter-efficient/free manner.

For better memory utilization, Schlag et al. (2021) use the delta rule to edit the memory dynamically. However, this is shown to underperform the gating mechanism (Mao, 2022), which is a classic method to erase irrelevant historical information in gated RNNs. Recently, Zhang et al. (2023) enforce orthogonality of memory vectors to potentially increase utiliziation.

Linear attention with decay or gates. Peng et al. (2021) use position-wise scalar gates for incorporating recency bias into linear attention, and has been revisited in recent work (Dao & Gu, 2024; Beck et al., 2024; Sun et al., 2024), while Mao (2022); Pramanik et al. (2023) use matrix-valued gates (obtained by the outer product) for more fine-grained memory control.

Scalar decays can be easily incorporated into chunkwise linear attention for training efficiency (Sun et al., 2023a; Qin et al., 2024a). With matrix-valued gates, the training efficiency becomes much more challenging. Both Mao (2022) and Katsch (2023)'s training algorithms involve materializing hidden states of all steps in HBM, which suffers from high I/O costs. Moreover, both approaches cannot take advantage of tensor cores. Our hardware-efficient training algorithm reduces or eliminates materialization and enables usage of tensor cores.

I/O-aware chunkwise linear attention. The chunkwise form of linear attention is well-known in the literature. Hua et al. (2022) first propose the chunkwise linear attention form, arguing that the training algorithm of Katharopoulos et al. (2020) is slow in practice. Sun et al. (2023a) and Qin et al. (2024a) generalize this form to linear attention with exponential decay (or ALiBi). Kacham et al. (2023); Lingle (2023) also derive similar chunkwise forms.

However, most chunkwise linear attention is not I/O-aware. To the best of our knowledge, only LIGHTNINGATTENTION2 (Qin et al., 2024a) (concurrent to our work) is I/O aware, and it is very similar to the non-materialization version of our FLASHLINEARATTENTION. We additionally propose a materialization version, which leverages sequence-level parallelism

and thus allows for higher training throughput at the cost of a slightly increasing memory footprint.

Other subquadratic models. Besides the Linear attention Transformer (Katharopoulos et al., 2020; Schlag et al., 2021) discussed in this work, previous studies have explored sparsifying attention with either a predefined fixed pattern (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020) or a context-aware learnable pattern (Roy et al., 2020; Kitaev et al., 2020; Ren et al., 2023) for sequence modeling with subquadratic complexity in the sequence length dimension. Leveraging convolutions for efficient sequence modeling has also been studied in works such as Dynamic Convolution (Wu et al., 2019), Long Convolution (Fu et al., 2023c; Qin et al., 2023a; Poli et al., 2023; Massaroli et al., 2023; Li et al., 2023d; Romero et al., 2021), and State Space Models (Gu et al., 2021a; Gupta & Berant, 2022; Gu et al., 2021b; Hasani et al., 2022; Smith et al., 2023; Ma et al., 2023).

A.2 Sequence parallelism

The chunk-wise parallel form of linear Transformers resembles the two-stage parallel prefix sum (or parallel scan) algorithm (Blelloch, 1990), which also combine chunk-wise parallel computations with inter-chunk communication (Chaurasia et al., 2015). It also resembles sequence parallelism used for accelerating attention-based Transformers (Li et al., 2023b), which has recently received much attention for long-sequence modeling (Liu et al., 2023; Li et al., 2023a; Brandon et al., 2023). Sequence-level parallelism also constitutes the main improvement of FlashAttention-2 (Dao, 2023) over FlashAttention-1 (Dao et al., 2022b). The main differences between these works are that (i) the chunk-level parallel form of linear Transformer needs only a single pass due to the linear complexity, while the sequence parallelism in Transformers needs L/C passes (i.e., left-to-right scan of key/value blocks for each query block) due to the inherent quadratic complexity, and (ii) the order of matrix multiplications is different. We also note that chunkwise linear attention could greatly reduce the communication cost between devices in the distributed training setting compared to softmax attention, which could open the door for extremely long sequence training.

Algorithm 2 FLASHLINEARATTENTION: Backward Pass

Input: $\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{O},\mathbf{dO} \in \mathbb{R}^{L\times d}$, chunk size $C \in [L]$, `materialize` $\in \{$`True`,`False`$\}$, $\mathbf{S} \in \mathbb{R}^{\frac{L}{C}\times d\times d}$ ▷ $\mathbf{S}$ is available when `materialize` is True
  Initialize $\mathbf{dS} = \mathbf{0} \in \mathbb{R}^{d\times d}$ on SRAM
  On chip, construct causal mask $\mathbf{M} \in \mathbb{R}^{C\times C}$
  if `materialize` **then** ▷ the materialization version
    for $n \leftarrow N, 1$ **do** ▷ in reverse order
      Store $\mathbf{dS}$ in HBM as $\mathbf{dS}_{[n]}$
      Load $\mathbf{Q}_{[n]}, \mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d}$ from HBM to SRAM.
      On chip, compute $\mathbf{dS} = \mathbf{dS} + \mathbf{Q}_{[n]}^\top \mathbf{dO}_{[n]}$
    end for
    parfor $n \leftarrow 1, N$ **do**
      Load $\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}, \mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d}$ from HBM to SRAM.
      Load $\mathbf{S}_{[n]}, \mathbf{dS}_{[n]} \in \mathbb{R}^{d\times d}$ from HBM to SRAM.
      On chip: $\mathbf{dQ} = \mathbf{dO}_{[n]}\mathbf{S}_{[n]}^\top + (\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^\top \odot \mathbf{M})\mathbf{K}_{[n]}$.
      On chip: $\mathbf{dK} = \mathbf{V}_{[n]}\mathbf{dS}_{[n]}^\top + (\mathbf{V}_{[n]}\mathbf{dO}_{[n]}^\top \odot \mathbf{M}^\top)\mathbf{Q}_{[n]}$
      On chip: $\mathbf{dV} = \mathbf{K}_{[n]}\mathbf{dS}_{[n]} + (\mathbf{Q}_{[n]}\mathbf{K}_{[n]}^\top \odot \mathbf{M})^\top \mathbf{dO}_{[n]}$
      Write $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$ to HBM as $\mathbf{dQ}_{[n]}, \mathbf{dK}_{[n]}, \mathbf{dV}_{[n]}$
    end parfor
  else ▷ the non-materialization version
    Initial $\mathbf{S} = \mathbf{0} \in \mathbb{R}^{d\times d}$ on SRAM
    for $n \leftarrow 1, N$ **do** ▷ hidden state recomputation
      Load $\mathbf{K}_{[n]}, \mathbf{V}_{[n]}, \mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d}$ from HBM to SRAM.
      On chip: $\mathbf{dQ} = \mathbf{dO}_{[n]}\mathbf{S}^\top + (\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^\top \odot \mathbf{M})\mathbf{K}_{[n]}$
      On chip: $\mathbf{S} = \mathbf{S} + \mathbf{K}_{[n]}^\top \mathbf{V}_{[n]}$
    end for
    for $n \leftarrow N, 1$ **do** ▷ in reverse order
      Load $\mathbf{Q}_{[n]}, \mathbf{K}_{[n]}, \mathbf{V}_{[n]}, \mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d}$ from HBM to SRAM.
      On chip, compute $\mathbf{dS} = \mathbf{dS} + \mathbf{Q}_{[n]}^\top \mathbf{dO}_{[n]}$
      On chip: $\mathbf{dQ} = \mathbf{dO}_{[n]}\mathbf{S}_{[n]}^\top + (\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^\top \odot \mathbf{M})\mathbf{K}_{[n]}$.
      On chip: $\mathbf{dK} = \mathbf{V}_{[n]}\mathbf{dS}_{[n]}^\top + (\mathbf{V}_{[n]}\mathbf{dO}_{[n]}^\top \odot \mathbf{M}^\top)\mathbf{Q}_{[n]}$
      On chip: $\mathbf{dV} = \mathbf{K}_{[n]}\mathbf{dS}_{[n]} + (\mathbf{Q}_{[n]}\mathbf{K}_{[n]}^\top \odot \mathbf{M})^\top \mathbf{dO}_{[n]}$
      Write $\mathbf{dQ}, \mathbf{dK}, \mathbf{dV}$ to HBM as $\mathbf{dQ}_{[n]}, \mathbf{dK}_{[n]}, \mathbf{dV}_{[n]}$
    end for
  end if
  return $\mathbf{dQ} = \{\mathbf{dQ}_{[1]} \dots \mathbf{dQ}_{[N]}\}, \mathbf{dK} = \{\mathbf{dK}_{[1]} \dots \mathbf{dK}_{[N]}\}, \mathbf{dV} = \{\mathbf{dV}_{[1]} \dots \mathbf{dV}_{[N]}\}$.

A.3 Hardware-ware algorithm

Many algorithms are fast in theory, but slow in practice, due to misalignment with hardware properties (Hooker, 2020; Saphra et al., 2023). For example, matmuls with butterfly matrices have theoretically lower complexity by using FFT, but in practice it is slow due to extensive memory transportation operations, motivating matrices (Dao et al., 2022a; Fu et al., 2023a) which can better align butterfly operators to GPUs. In practice it is important to reduce HBM I/O cost using techniques such as tiling and recomputation and leverage tensor cores as much as possible. Our FLASHLINEARATTENTION is similar in spirit to FLASHATTENTION (Dao et al., 2022b; Dao, 2023) and FLASHCONVFFT (Fu et al., 2023d), which implement I/O-aware versions of neural network layers to enable practical wallclock speedups. Concurrent work by Qin et al. (2024a) also proposes an I/O-aware version of linear attention, which is similar to the non-materialization version of FLASHLINEARATTENTION. We additionally propose a materialization version, which leverages sequence-level parallelism and thus allows for higher training throughput at the cost of a slightly increasing memory footprint.

B Details for Chunkwise (Gated) Linear Attention

Backward pass of FLASHLINEARATTENTION. The pseduocode for backward pass of linear attention is listed in Algorithm 2.

Pseudo codes of GLA. We first present the direct adaptions of FLASHLINEARATTENTION to training GLA without secondary-level chunking. Specifically, Alg. 3 and 4 shows the forward/backward pass for the materialization version; Alg. 5 and 6 for the non-materialization version. We show the psuedo code of our secondary-level chunking in Pytorch style in Listing 1.

```
def gated_linear_attention_forward(Q, K, V, a, C, c):
    '''
    Q/K/V: query/key/value
    a: log forget gate
    C/c: chunk size, subchunk size
    '''
    # L: sequence length, d: head dimension
    L, d_k = Q.shape
    d_v = V.shape[-1]
    S = torch.zeros(d_k, d_v)
    O = torch.empty_like(V)
    # cumsum of log decay within a chunk
    B = torch.empty_like(a)
    # local compute of cumulative product of decay within a chunk
    for i in range(0, L//C):
        b = torch.zeros(d_k)
        for j in range(0, C):
            b += a[i]
            B[i] = b

    for i in range(0, L // C):
        r = range(i*C,(i+1)*C)
        # (C, d) chunking
        bq, bk, bv, bb = Q[r], K[r], V[r], B[r]
        b = bb[-1,None]
        #inter-chunk w/ matmul
        q, k, g = bq*(bb.exp()), bk*((b-bb).exp()), b.exp()
        o = q @ S
        #hidden state update
        S = g.t() * S + k.t() @ bv
        #intra-chunk (secondary chunking)
        for j in range(0, C // c):
            t = range(j*c, (j+1)*c)
            #(c, head_dim) subchunking
            q, k, v, b = bq[t], bk[t], bv[t], bb[t]
            p = torch.zeros(c,c)
            #intra-subchunk w/o matmul.
            for m in range(c):
```

```
                for n in range(m+1):
                    p[m,n]=torch.sum(q[m]*k[n]*((b[m]-b[n]).exp()))
            o[t] += p @ v
            # inter-subchunk w/ matmul
            z = b[0, None]
            q = q * (b-z).exp()
            for u in range(0, j):
                y = range(u*c, (u+1)*c)
                p = q @ (bk[y]*(z-bb[y]).exp()).t()
                o[t] += p@bv[y]
        O[r] = o
    return O
```

Listing 1: Pytorch-like code snippet of our two-level chunking algorithm for training GLA. We omit the dimensions of batch size and number of heads for clarity

Derivations of $\mathbf{d}\log\boldsymbol{\alpha}_t$. We show the derivations for the following gradient form.

$$\mathbf{d}\log\boldsymbol{b}_t = \boldsymbol{k}_t \odot \mathbf{d}\boldsymbol{k}_t - \boldsymbol{q}_t \odot \mathbf{d}\boldsymbol{q}_t,$$
$$\mathbf{d}\log\boldsymbol{\alpha}_t = \sum_{t \le i \le L} \mathbf{d}\log\boldsymbol{b}_i.$$

By unrolling the recurrence, we have

$$\begin{aligned}\boldsymbol{o}_t = \boldsymbol{q}_t \mathbf{S}_t &= \sum_{i=1}^{t} (\boldsymbol{q}_t \odot \boldsymbol{b}_t) \left(\frac{\boldsymbol{k}_i}{\boldsymbol{b}_i}\right)^\top \boldsymbol{v}_i \\ &= \sum_{i=1}^{t} (\boldsymbol{q}_t \odot \exp(\log \boldsymbol{b}_t))(\boldsymbol{k}_i \odot \exp(-\log \boldsymbol{b}_i))^\top \boldsymbol{v}_i\end{aligned}$$

Algorithm 3 Forward pass for gated linear attention (w. materialization)

Input: $\mathbf{Q},\mathbf{K},\in\mathbb{R}^{L\times d_k}, \mathbf{V}\in\mathbb{R}^{L\times d_v}, \mathbf{G}=[\boldsymbol{\alpha}_1 ... \boldsymbol{\alpha}_L]\in\mathbb{R}^{L\times d_k}$, chunk size C
Divide $\mathbf{Q},\mathbf{K},\mathbf{G}$ into $N=\frac{L}{C}$ blocks $\{\mathbf{Q}_{[1]}...\mathbf{Q}_{[N]}\}, \{\mathbf{K}_{[1]}...\mathbf{K}_{[N]}\}, \{\mathbf{G}_{[1]}...\mathbf{G}_{[N]}\}$ of size $C\times d_k$ each.
Divide $\mathbf{V}$ into N blocks $\{\mathbf{V}_{[1]}...\mathbf{V}_{[N]}\}$ of size $C\times d_v$ each.
Initialize $\mathbf{S}=\mathbf{0}\in\mathbb{R}^{d_k\times d_v}$ on SRAM
for $n\leftarrow 1,N$ **do**
  Write $\mathbf{S}$ to HBM as $\mathbf{S}_{[n]}$.
  Load $\mathbf{K}_{[n]},\mathbf{G}_{[n]}\in\mathbb{R}^{C\times d_k}$ from HBM to SRAM.
  Load $\mathbf{V}_{[n]}\in\mathbb{R}^{C\times d_v}$ from HBM to SRAM.
  On chip, compute $\boldsymbol{\gamma}_{[n]}\in\mathbb{R}^{d_k}, \boldsymbol{\Gamma}_{[n]}\in\mathbb{R}^{C\times d_k}$ and $\tilde{\mathbf{K}}_{[n]}=\mathbf{K}_{[n]}\odot\boldsymbol{\Gamma}_{[n]}$.
  On chip, compute $\mathbf{S}=\left(\boldsymbol{\gamma}_{[n]}^\top\mathbf{1}\right)\odot\mathbf{S}+\tilde{\mathbf{K}}_{[n]}^\top\mathbf{V}_{[n]}$.
end for
parfor $n\leftarrow 1,N$ **do**
  Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{G}_{[n]}\in\mathbb{R}^{C\times d_k}$ from HBM to SRAM.
  Load $\mathbf{V}_{[n]}\in\mathbb{R}^{C\times d_v}$ from HBM to SRAM.
  Load $\mathbf{S}_{[n]}\in\mathbb{R}^{d_k\times d_v}$ from HBM to SRAM.
  On chip, construct causal mask $\mathbf{M}\in\mathbb{R}^{C\times C}$
  On chip, compute $\boldsymbol{\Lambda}_{[n]},\boldsymbol{\Gamma}_{[n]}\in\mathbb{R}^{C\times d_k}$
  On chip, compute $\tilde{\mathbf{Q}}_{[n]}=\mathbf{Q}_{[n]}\odot\boldsymbol{\Lambda}_{[n]}, \tilde{\mathbf{K}}_{[n]}=\mathbf{K}_{[n]}\odot\boldsymbol{\Gamma}_{[n]}, \bar{\mathbf{K}}_{[n]}=\mathbf{K}_{[n]}/\boldsymbol{\Lambda}_{[n]}$
  On chip, compute $\mathbf{O}_{[n]}^{\text{inter}}=\tilde{\mathbf{Q}}_{[n]}\mathbf{S}_{[n]}\in\mathbb{R}^{C\times d_v}$
  On chip, compute $\mathbf{P}=(\tilde{\mathbf{Q}}_{[n]}\bar{\mathbf{K}}_{[n]}^\top)\odot\mathbf{M}\in\mathbb{R}^{C\times C}$
  On chip, compute $\mathbf{O}^{\text{intra}}=\mathbf{P}\mathbf{V}_{[n]}$
  On chip, compute $\mathbf{O}_{[n]}=\mathbf{O}^{\text{inter}}+\mathbf{O}^{\text{intra}}$
  Store $\mathbf{O}_{[n]}$ to HBM.
end parfor
return $\mathbf{O}=\{\mathbf{O}_{[1]}...\mathbf{O}_{[N]}\}, \mathbf{S}=\{\mathbf{S}_{[1]}...\mathbf{S}_{[N]}\}$.

Algorithm 4 Backward pass for gated linear attention (w. materialization)

Input: $\mathbf{Q},\mathbf{K},\mathbf{G} \in \mathbb{R}^{L\times d_k}$, $\mathbf{V},\mathbf{O},\mathbf{dO} \in \mathbb{R}^{L\times d_v}$, chunk size C
Initialize $\mathbf{dS} = \mathbf{0} \in \mathbb{R}^{d_k\times d_v}$ on SRAM
for $n \leftarrow N,1$ **do**
Store $\mathbf{dS}$ in HBM as $\mathbf{dS}_{[n]}$
Load $\mathbf{G}_{[n]} \in \mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{Q}_{[n]} \in \mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d_v}$ from HBM to SRAM.
On chip, compute $\boldsymbol{\gamma}_{[n]}$, $\boldsymbol{\Gamma}_{[n]}$ and $\tilde{\mathbf{Q}}[n] = \mathbf{Q}_{[n]} \odot \boldsymbol{\Gamma}[n]$
On chip, compute $\mathbf{dS} = \left(\boldsymbol{\gamma}_{[n]}^{\top}\mathbf{1}\right) \odot \mathbf{dS} + \tilde{\mathbf{Q}}_{[n]}^{\top}\mathbf{dO}_{[n]}$
end for
parfor $n \leftarrow 1,N$ **do**
Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{G}_{[n]} \in \mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{S}_{[n]} \in \mathbb{R}^{d_k\times d_v}$ from HBM to SRAM.
Load $\mathbf{V}_{[n]},\mathbf{O}_{[n]},\mathbf{dO}_{[n]} \in \mathbb{R}^{C\times d_v}$ from HBM to SRAM.
Load $\mathbf{dS}_{[n]} \in \mathbb{R}^{d_k\times d_v}$ from HBM to SRAM.
On chip, construct causal mask $\mathbf{M} \in \mathbb{R}^{B\times B}$
On chip, compute $\boldsymbol{\Lambda}_{[n]},\boldsymbol{\Gamma}_{[n]} \in \mathbb{R}^{C\times d_k}$
On chip, compute $\tilde{\mathbf{Q}}_{[n]} = \mathbf{Q}_{[n]} \odot \boldsymbol{\Lambda}_{[n]}$, $\tilde{\mathbf{K}}_{[n]} = \mathbf{K}_{[n]} \odot \boldsymbol{\Gamma}_{[n]}$, $\bar{\mathbf{K}}_{[n]} = \mathbf{K}_{[n]} / \boldsymbol{\Lambda}_{[n]}$.
On chip, compute $\mathbf{P} = (\tilde{\mathbf{Q}}_{[n]}\tilde{\mathbf{K}}_{[n]}^{\top}) \odot \mathbf{M} \in \mathbb{R}^{C\times C}$
On chip, compute $\mathbf{dP} = (\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^{\top}) \odot \mathbf{M}$
On chip, compute $\mathbf{d}\bar{\mathbf{K}}_{[n]} = \tilde{\mathbf{Q}}_{[n]}\mathbf{dP}^{\top}$
On chip, compute $\mathbf{d}\tilde{\mathbf{K}}_{[n]} = \mathbf{V}_{[n]}\mathbf{dS}_{[n]}^{\top}$
On chip, compute $\mathbf{dK}_{[n]} = \mathbf{d}\tilde{\mathbf{K}}_{[n]} \odot \boldsymbol{\Gamma}_{[n]} + \mathbf{d}\bar{\mathbf{K}}_{[n]} / \boldsymbol{\Lambda}_{[n]}$
On chip, compute $\mathbf{d}\tilde{\mathbf{Q}}_{[n]} = \mathbf{dP}\bar{\mathbf{K}}_{[n]} + \mathbf{dO}_{[n]}\mathbf{S}_{[n]}^{\top}$
On chip, compute $\mathbf{dQ}_{[n]} = \mathbf{d}\tilde{\mathbf{Q}}_{[n]} \odot \boldsymbol{\Lambda}_{[n]}$
On chip, compute $\mathbf{dV}_{[n]} = \mathbf{P}^{\top}\mathbf{dO}_{[n]} + \tilde{\mathbf{K}}_{[n]}\mathbf{dS}_{[n]}$
Store $\mathbf{dK}_{[n]},\mathbf{dV}_{[n]}$ in HBM.
end parfor
Let $\mathbf{dQ} = \{\mathbf{dQ}_{[1]}...\mathbf{dQ}_{[N]}\}$, $\mathbf{dK} = \{\mathbf{dK}_{[1]}...\mathbf{dK}_{[N]}\}$, $\mathbf{dV} = \{\mathbf{dV}_{[1]}...\mathbf{dV}_{[N]}\}$.
Compute $\mathbf{dA} = \mathbf{Q} \odot \mathbf{dQ} - \mathbf{K} \odot \mathbf{dK}$, $\mathbf{dG} = \text{revcum}(\mathbf{dA})$
return $\mathbf{dQ},\mathbf{dK},\mathbf{dV},\mathbf{dG}$

Algorithm 5 Forward pass for gated linear attention (w/o. materialization)

Input: $\mathbf{Q},\mathbf{K},\in \mathbb{R}^{L\times d_k}$, $\mathbf{V} \in \mathbb{R}^{L\times d_v}$, $\mathbf{G} = [\boldsymbol{\alpha}_1...\boldsymbol{\alpha}_L] \in \mathbb{R}^{L\times d_k}$, chunk size C
Divide $\mathbf{Q},\mathbf{K},\mathbf{G}$ into $N = \frac{L}{B}$ blocks $\{\mathbf{Q}_{[1]}...\mathbf{Q}_{[N]}\}$, $\{\mathbf{K}_{[1]}...\mathbf{K}_{[N]}\}$, $\{\mathbf{G}_{[1]}...\mathbf{G}_{[N]}\}$ of size $C \times d_k$ each.
Divide $\mathbf{V}$ into N blocks $\{\mathbf{V}_{[1]}...\mathbf{V}_{[N]}\}$ of size $C \times d_v$ each.
Initialize $\mathbf{S} = \mathbf{0} \in \mathbb{R}^{d_k\times d_v}$ on SRAM
for $n \leftarrow 1,N$ **do**
Write $\mathbf{S}$ to HBM as $\mathbf{S}_{[n]}$.
Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{G}_{[n]} \in \mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{V}_{[n]} \in \mathbb{R}^{C\times d_v}$ from HBM to SRAM.
On chip, compute $\boldsymbol{\gamma}_{[n]} \in \mathbb{R}^{d_k}$, $\boldsymbol{\Gamma}_{[n]} \in \mathbb{R}^{C\times d_k}$ and $\tilde{\mathbf{K}}_{[n]} = \mathbf{K}_{[n]} \odot \boldsymbol{\Gamma}_{[n]}$.
On chip, construct causal mask $\mathbf{M} \in \mathbb{R}^{C\times C}$
On chip, compute $\boldsymbol{\Lambda}_{[n]},\boldsymbol{\Gamma}_{[n]} \in \mathbb{R}^{C\times d_k}$
On chip, compute $\tilde{\mathbf{Q}}_{[n]} = \mathbf{Q}_{[n]} \odot \boldsymbol{\Lambda}_{[n]}$, $\tilde{\mathbf{K}}_{[n]} = \mathbf{K}_{[n]} \odot \boldsymbol{\Gamma}_{[n]}$, $\bar{\mathbf{K}}_{[n]} = \mathbf{K}_{[n]} / \boldsymbol{\Lambda}_{[n]}$.
On chip, compute $\mathbf{O}_{[n]}^{\text{inter}} = \tilde{\mathbf{Q}}_{[n]}\mathbf{S}_{[n]} \in \mathbb{R}^{C\times d_v}$
On chip, compute $\mathbf{P} = (\tilde{\mathbf{Q}}_{[n]}\bar{\mathbf{K}}_{[n]}^{\top}) \odot \mathbf{M} \in \mathbb{R}^{C\times C}$
On chip, compute $\mathbf{O}^{\text{intra}} = \mathbf{P}\mathbf{V}_{[n]}$
On chip, compute $\mathbf{O}_{[n]} = \mathbf{O}^{\text{inter}} + \mathbf{O}^{\text{intra}}$
Store $\mathbf{O}_{[n]}$ to HBM.
On chip, compute $\mathbf{S} = \left(\boldsymbol{\gamma}_{[n]}^{\top}\mathbf{1}\right) \odot \mathbf{S} + \tilde{\mathbf{K}}_{[n]}^{\top}\mathbf{V}_{[n]}$.
end for
return $\mathbf{O} = \{\mathbf{O}_{[1]}...\mathbf{O}_{[N]}\}$.

Algorithm 6 Backward pass for gated linear attention (w/o. materialization)

Input: $\mathbf{Q},\mathbf{K},\mathbf{G}\in\mathbb{R}^{L\times d_k}$, $\mathbf{V},\mathbf{O},\mathbf{dO}\in\mathbb{R}^{L\times d_v}$, chunk size C
Initialize $\mathbf{S}=\mathbf{0}\in\mathbb{R}^{d_k\times d_v}$ on SRAM
for $n\leftarrow 1,N$ **do**
Load $\mathbf{G}_{[n]}\in\mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{Q}_{[n]}\in\mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{dO}_{[n]}\in\mathbb{R}^{C\times d_v}$ from HBM to SRAM.
On chip, compute $\boldsymbol{\gamma}_{[n]}\in\mathbb{R}^{d_k},\boldsymbol{\Gamma}_{[n]}\in\mathbb{R}^{C\times d_k}$ and $\tilde{\mathbf{K}}_{[n]}=\mathbf{K}_{[n]}\odot\boldsymbol{\Gamma}_{[n]}$.
On chip, compute $\mathbf{dP}=\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^{\top}$
On chip, compute $\mathbf{d}\tilde{\mathbf{Q}}_{[n]}=\mathbf{dP}\tilde{\mathbf{K}}_{[n]}+\mathbf{dO}_{[n]}\mathbf{S}^{\top}$
On chip, compute $\mathbf{dQ}=\mathbf{d}\tilde{\mathbf{Q}}_{[n]}\odot\boldsymbol{\Gamma}[n]$
Store $\mathbf{dQ}_{[n]}$ to HBM.
On chip, compute $\mathbf{S}=\left(\boldsymbol{\gamma}_{[n]}^{\top}\mathbf{1}\right)\odot\mathbf{S}+\tilde{\mathbf{K}}_{[n]}^{\top}\mathbf{V}_{[n]}$.
end for
Initialize $\mathbf{dS}=\mathbf{0}\in\mathbb{R}^{d_k\times d_v}$ on SRAM
for $n\leftarrow N,1$ **do**
Load $\mathbf{Q}_{[n]},\mathbf{K}_{[n]},\mathbf{G}_{[n]}\in\mathbb{R}^{C\times d_k}$ from HBM to SRAM.
Load $\mathbf{V}_{[n]},\mathbf{O}_{[n]},\mathbf{dO}_{[n]}\in\mathbb{R}^{C\times d_v}$ from HBM to SRAM.
On chip, construct causal mask $\mathbf{M}\in\mathbb{R}^{C\times C}$
On chip, compute $\boldsymbol{\Lambda}_{[n]},\boldsymbol{\Gamma}_{[n]}\in\mathbb{R}^{C\times d_k}$
On chip, compute $\tilde{\mathbf{Q}}_{[n]}=\mathbf{Q}_{[n]}\odot\boldsymbol{\Lambda}_{[n]}$, $\tilde{\mathbf{K}}_{[n]}=\mathbf{K}_{[n]}\odot\boldsymbol{\Gamma}_{[n]}$.
On chip, compute $\mathbf{P}=(\tilde{\mathbf{Q}}_{[n]}\tilde{\mathbf{K}}_{[n]}^{\top})\odot\mathbf{M}\in\mathbb{R}^{C\times C}$
On chip, compute $\mathbf{dP}=(\mathbf{dO}_{[n]}\mathbf{V}_{[n]}^{\top})\odot\mathbf{M}$
On chip, compute $\mathbf{d}\bar{\mathbf{K}}_{[n]}=\tilde{\mathbf{Q}}_{[n]}\mathbf{dP}^{\top}$
On chip, compute $\mathbf{d}\tilde{\mathbf{K}}_{[n]}=\mathbf{V}_{[n]}\mathbf{dS}_{[n]}^{\top}$
On chip, compute $\mathbf{dK}_{[n]}=\mathbf{d}\tilde{\mathbf{K}}_{[n]}\odot\boldsymbol{\Gamma}_{[n]}+\mathbf{d}\bar{\mathbf{K}}_{[n]}/\boldsymbol{\Lambda}_{[n]}$
On chip, compute $\mathbf{dV}_{[n]}=\mathbf{P}^{\top}\mathbf{dO}_{[n]}+\tilde{\mathbf{K}}_{[n]}\mathbf{dS}$
Store $\mathbf{dQ}_{[n]},\mathbf{dK}_{[n]},\mathbf{dV}_{[n]}$ in HBM.
On chip, compute $\mathbf{dS}=\left(\boldsymbol{\gamma}_{[n]}^{\top}\mathbf{1}\right)\odot\mathbf{dS}+\tilde{\mathbf{Q}}_{[n]}^{\top}\mathbf{dO}_{[n]}$
end for
Let $\mathbf{dQ}=\{\mathbf{dQ}_{[1]}...\mathbf{dQ}_{[N]}\}$, $\mathbf{dK}=\{\mathbf{dK}_{[1]}...\mathbf{dK}_{[N]}\}$, $\mathbf{dV}=\{\mathbf{dV}_{[1]}...\mathbf{dV}_{[N]}\}$.
Compute $\mathbf{dA}=\mathbf{Q}\odot\mathbf{dQ}-\mathbf{K}\odot\mathbf{dK}$, $\mathbf{dG}=\text{revcum}(\mathbf{dA})$
return $\mathbf{dQ},\mathbf{dK},\mathbf{dV},\mathbf{dG}$

where at the second step, we apply a trivial identity: $\exp(\log x)=x$. We first derive the gradients wrt. query/key vectors,

$$\mathbf{d}\boldsymbol{q}_t=\sum_{i=1}^{t}\langle\mathbf{d}\boldsymbol{o}_t,\boldsymbol{v}_i\rangle\boldsymbol{b}_t\odot\boldsymbol{k}_i/\boldsymbol{b}_i,$$
$$\mathbf{d}\boldsymbol{k}_i=\sum_{t=i}^{L}\langle\mathbf{d}\boldsymbol{o}_t,\boldsymbol{v}_i\rangle\boldsymbol{q}_t\odot\boldsymbol{b}_t/\boldsymbol{b}_i.$$

Then for the gradients wrt. the logits of the accumulative gates,

$$\mathbf{d}\log\boldsymbol{b}_t=\underbrace{\boldsymbol{q}_t\odot\sum_{i=1}^{t}\langle\mathbf{d}\boldsymbol{o}_t,\boldsymbol{v}_i\rangle\odot\boldsymbol{b}_t\odot\boldsymbol{k}_i/\boldsymbol{b}_i}_{\mathbf{d}\boldsymbol{q}_t}-\underbrace{\boldsymbol{k}_t\odot\sum_{i=t}^{L}\langle\mathbf{d}\boldsymbol{o}_i,\boldsymbol{v}_t\rangle\boldsymbol{q}_i\odot\boldsymbol{b}_i/\boldsymbol{b}_t}_{\mathbf{d}\boldsymbol{k}_t}.$$

where we change the index notation for the $\mathbf{d}\boldsymbol{k}$ term. It now becomes clear that

$$\mathbf{d}\log\boldsymbol{b}_t=\boldsymbol{q}_t\odot\mathbf{d}\boldsymbol{q}_t-\boldsymbol{k}_t\odot\mathbf{d}\boldsymbol{k}_t.$$

Since $\log\boldsymbol{b}_t=\sum_{i=1}^{t}\log\boldsymbol{\alpha}_i$, we get $\mathbf{d}\log\boldsymbol{\alpha}_t=\sum_{t=i}^{L}\mathbf{d}\log\boldsymbol{b}_i$.

C General Gated Linear Attention

In the main paper, we use a simplified parameterization where $\boldsymbol{\beta}$ is fixed to $\mathbf{1}$ in the following gated linear attention.

$$\mathbf{S}_t = (\boldsymbol{\alpha}_t^\top \boldsymbol{\beta}_t) \odot \mathbf{S}_{t-1} + \boldsymbol{k}_t^\top \boldsymbol{v}_t,$$

Though empirically we found that making $\boldsymbol{\beta}$ learnable does not lead to performance gain, we show here that the general form still enjoys parallel form and chunk-wise form, which could be potentially useful for future development of linear attention models.

C.1 Parallel form

By unrolling the recurrence we have,

$$\boldsymbol{o}_t = \boldsymbol{q}_t \mathbf{S}_t = \boldsymbol{q}_t \sum_{i=1}^{t} \big((\prod_{i+1}^{t} \mathbf{G}_i) \odot (\boldsymbol{k}_i^\top \boldsymbol{v}_i) \big) \tag{5}$$

By taking advantage of the mixed product property of Kronercker/outer product, we have

$$(\prod_{j=i+1}^{t} \mathbf{G}_j) \odot (\boldsymbol{k}_i^\top \boldsymbol{v}_i) = \big((\frac{\boldsymbol{b}_t}{\boldsymbol{b}_i})^\top (\frac{\boldsymbol{d}_t}{\boldsymbol{d}_i}) \big) \odot (\boldsymbol{k}_i^\top \boldsymbol{v}_i) \tag{6}$$

$$= \left(\frac{\boldsymbol{b}_t}{\boldsymbol{b}_i} \odot \boldsymbol{k}_i \right)^\top \left(\frac{\boldsymbol{d}_t}{\boldsymbol{d}_i} \odot \boldsymbol{v}_i \right) \tag{7}$$

where $\boldsymbol{b}_t = \prod_{j=1}^{t} \boldsymbol{\alpha}_j, \boldsymbol{d}_t = \prod_{j=1}^{t} \boldsymbol{\beta}_j$. By plugging it into the expanded recurrence, we have the following form.

$$\boldsymbol{o}_t = \boldsymbol{q}_t \mathbf{S}_t = \boldsymbol{q}_t \sum_{i=1}^{t} \big((\prod_{i+1}^{t} \mathbf{G}_i) \odot (\boldsymbol{k}_i^\top \boldsymbol{v}_i) \big) \tag{8}$$

$$= \boldsymbol{q}_t \sum_{i=1}^{t} \left(\frac{\boldsymbol{b}_t}{\boldsymbol{b}_i} \odot \boldsymbol{k}_i \right)^\top \left(\frac{\boldsymbol{d}_t}{\mathbf{B}_i} \odot \boldsymbol{v}_i \right) \tag{9}$$

$$= \sum_{i=1}^{t} \left(\boldsymbol{q}_t \left(\frac{\boldsymbol{b}_t}{\boldsymbol{b}_i} \odot \boldsymbol{k}_i \right)^\top \right) \left(\frac{\boldsymbol{d}_t}{\boldsymbol{d}_i} \odot \boldsymbol{v}_i \right) \tag{10}$$

$$= \sum_{i=1}^{t} \underbrace{\left\langle \boldsymbol{q}_t, \frac{\boldsymbol{b}_t}{\boldsymbol{b}_i} \odot \boldsymbol{k}_t \right\rangle}_{\mathbb{R}^{1\times 1}} \underbrace{\left(\frac{\boldsymbol{d}_t}{\boldsymbol{d}_i} \odot \boldsymbol{v}_t \right)}_{\mathbb{R}^{1\times d_v}} \tag{11}$$

$$= \sum_{i=1}^{t} \left(\left\langle \boldsymbol{q}_t \odot \boldsymbol{b}_t, \frac{\boldsymbol{k}_i}{\boldsymbol{b}_i} \right\rangle \frac{\boldsymbol{v}_i}{\boldsymbol{d}_i} \right) \odot \boldsymbol{d}_t \tag{12}$$

$$= \sum_{i=1}^{t} \left((\boldsymbol{q}_t \odot \boldsymbol{b}_t) \left(\frac{\boldsymbol{k}_i}{\boldsymbol{b}_i} \right)^\top \left(\frac{\boldsymbol{v}_i}{\boldsymbol{d}_i} \right) \right) \odot \boldsymbol{d}_t \quad \in \mathbb{R}^{1\times d_v} \tag{13}$$

Eq. 10 is by linearity and associative property of matrix multiplication, Eq. 12 is derived based on $\langle \boldsymbol{a}, \boldsymbol{b} \odot \boldsymbol{c} \rangle = \langle \boldsymbol{a} \odot \boldsymbol{b}, \boldsymbol{c} \rangle$. The final form has following equivalent parallel form similar to the parallel form of linear/softmax attention.

$$\tilde{\mathbf{Q}} = \mathbf{Q} \odot \mathbf{B} \qquad \tilde{\mathbf{K}} = \mathbf{K} / \mathbf{B} \qquad \tilde{\mathbf{V}} = \mathbf{V} / \mathbf{D} \tag{14}$$

$$\tilde{\mathbf{O}} = (\tilde{\mathbf{Q}} \tilde{\mathbf{K}}^\top \odot \mathbf{M}) \tilde{\mathbf{V}} \qquad \mathbf{O} = \tilde{\mathbf{O}} \odot \mathbf{D} \tag{15}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{B} \in \mathbb{R}^{L\times d_k}$, $\mathbf{V}, \mathbf{D} \in \mathbb{R}^{L\times d_v}$, $\mathbf{M} \in \mathbb{R}^{L\times L}$ denotes the causal mask.

C.2 Chunkwise parallel form

Now we show that the chunkwise parallel form for efficient training of general linear attention. Suppose $\mathbf{X}$ is now split into $\frac{L}{C}$ chunks, each of length C. Let $\mathbf{S}_{[i]} \in \mathbb{R}^{d_k \times d_v}$ be the chunk-level hidden state after processing i chunks, i.e., $\mathbf{S}_{[i]} := \mathbf{S}_{iC}$. Further

let $\mathbf{K}_{[i+1]} := \mathbf{K}_{iC+1:(i+1)C} \in \mathbb{R}^{C\times d_k}$, $\mathbf{V}_{[i+1]} := \mathbf{V}_{iC+1:(i+1)C} \in \mathbb{R}^{C\times d_v}$. The inter-chunk recurrence is then given by,

$$\mathbf{S}_{[i+1]} = \left(\left(\frac{\mathbf{B}_{(i+1)C}}{\mathbf{B}_{iC}}\right)^{\top}\left(\frac{\mathbf{D}_{(i+1)C}}{\mathbf{D}_{iC}}\right)\right)\odot\mathbf{S}_{[i]} + \left(\mathbf{B}'_{[i+1]}\odot\mathbf{K}_{[i+1]}\right)^{\top}\left(\mathbf{D}'_{[i+1]}\odot\mathbf{V}_{[i+1]}\right),$$

where $(\mathbf{B}'_{[i+1]})_j = \frac{\mathbf{B}_{(i+1)C}}{\mathbf{B}_{iC+j}} \in \mathbb{R}^{1\times d_k}$ and $(\mathbf{D}'_{[i+1]})_j = \frac{\mathbf{D}_{(i+1)C}}{\mathbf{D}_{iC+j}} \in \mathbb{R}^{1\times d_v}$ for $j\in[1,C]$, $i\in[0,L/C-1]$. (Therefore we have $\mathbf{B}'_{[i+1]}\in\mathbb{R}^{C\times d_k}$,$\mathbf{D}'_{[i+1]}\in\mathbb{R}^{C\times d_v}$.) The intra-chunk parallel computation is then given by,

$$\tilde{\mathbf{O}}_{[i+1]} = \underbrace{\left((\mathbf{Q}_{[i+1]}\odot\mathbf{B}^{\dagger}_{[i+1]})\mathbf{S}_{[i]}\right)\odot\mathbf{D}^{\dagger}_{[i+1]}}_{\text{inter-chunk}} + \underbrace{(\tilde{\mathbf{Q}}_{[i+1]}\tilde{\mathbf{K}}^{\top}_{[i+1]}\odot\mathbf{M})\tilde{\mathbf{V}}_{[i+1]}}_{\text{intra-chunk}}, \tag{16}$$

$$\mathbf{O}_{[i+1]} = \tilde{\mathbf{O}}_{[i+1]}/\mathbf{D}^{\dagger}_{[i+1]}, \tag{17}$$

where $(\mathbf{B}^{\dagger}_{[i+1]})_j = \frac{\mathbf{B}_{iC+j}}{\mathbf{B}_{iC}} \in \mathbb{R}^{1\times d_k}$ and $(\mathbf{D}^{\dagger}_{[i+1]})_j = \frac{\mathbf{D}_{iC+j}}{\mathbf{D}_{iC}} \in \mathbb{R}^{1\times d_v}$ for $j\in[1,C]$, $i\in[0,L/C-1]$. Subsequently, we have $\tilde{\mathbf{Q}}_{[i+1]} = \mathbf{Q}_{[i+1]}\odot\mathbf{B}^{\dagger}_{[i+1]}$,$\tilde{\mathbf{K}}_{[i+1]} = \frac{\mathbf{K}_{[i+1]}}{\mathbf{B}^{\dagger}_{[i+1]}}$,$\tilde{\mathbf{V}}_{[i+1]} = \mathbf{V}_{[i+1]}\odot\mathbf{D}^{\dagger}_{[i+1]}$. For initial values, we set $\mathbf{S}_0 = \mathbf{0}$, $\mathbf{B}_0 = \mathbf{1}$, $\mathbf{D}_0 = \mathbf{1}$. Intuitively, $\mathbf{B}'_{[i]}$ encodes the cumulative decay from the start of a chunk which will be used to propagate the hidden states from the previous chunk $\mathbf{S}_{[i]}$; $\mathbf{B}^{\dagger}_{[i]}$ encodes the decay to the end of a chunk which will be used to accumulate information to be added to the next hidden state $\mathbf{S}_{[i+1]}$.

The chunkwise form given here is a generalization of several existing forms for linear attention. If we set $\mathbf{A}_{ij}=1$, $\mathbf{B}_{ij}=1$, it reduces to the chunk-wise form presented in the main paper for vanilla linear attention; if we set $\mathbf{A}_{ij}=1$, $\mathbf{B}_{ij}=\gamma^{i+1}$, it becomes RetNet's chunk-wise form (Sun et al., 2023a). As such, our formulation can be regarded as a generalized chunk-wise parallel form for linear attention that enables fine-grained data-dependent decay.

Memory-efficient computation of $\mathbf{d}\boldsymbol{\alpha}$ and $\mathbf{d}\boldsymbol{\beta}$ In the general form, we show that the gradient wrt. $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ admits the following closed form, which allows computing $\mathbf{d}\boldsymbol{\alpha}$ and $\mathbf{d}\boldsymbol{\beta}$ without instantiating $\mathbf{S}$ in HBM.

$$\begin{aligned}
\mathbf{d}\log\boldsymbol{b}_t &= \boldsymbol{k}_t\odot\mathbf{d}\boldsymbol{k}_t - \boldsymbol{q}_t\odot\mathbf{d}\boldsymbol{q}_t,\\
\mathbf{d}\log\boldsymbol{\alpha}_t &= \sum_{t\le i\le L}\mathbf{d}\log\boldsymbol{b}_i\\
\mathbf{d}\log\boldsymbol{d}_t &= \boldsymbol{o}_t\odot\mathbf{d}\boldsymbol{o}_t - \boldsymbol{v}_t\odot\mathbf{d}\boldsymbol{v}_t,\\
\mathbf{d}\log\boldsymbol{\beta}_t &= \sum_{t\le i\le L}\mathbf{d}\log\boldsymbol{d}_i.
\end{aligned}$$

where $\log\boldsymbol{b}_t = \sum_{i=1}^{t}\log\boldsymbol{\alpha}_i$, $\log\boldsymbol{b}=\boldsymbol{d}_t = \sum_{i=1}^{t}\boldsymbol{\beta}_i$ (or alternatively $\boldsymbol{b}_t = \prod_{i=1}^{t}\boldsymbol{\alpha}_i$, $\boldsymbol{d}_t = \prod_{i=1}^{t}\boldsymbol{\beta}_i$). We apply the trick to compute $\mathbf{d}\log\boldsymbol{b}_t$ and $\mathbf{d}\log\boldsymbol{d}_t$ for the following cumulative-sum form.

$$\boldsymbol{o}_t = \sum_{i=1}^{t}\left((\boldsymbol{q}_t\odot\boldsymbol{b}_t)\left(\frac{\boldsymbol{k}_i}{\boldsymbol{b}_i}\right)^{\top}\left(\frac{\boldsymbol{v}_i}{\boldsymbol{d}_i}\right)\right)\odot\boldsymbol{d}_t \quad \in\mathbb{R}^{1\times d_v}.$$

The gradient of $\log\boldsymbol{b}_t$ comes from two sources: one associated with $\boldsymbol{q}_t$, the other associated with $\boldsymbol{k}_i$. Similarly, $\log\boldsymbol{b}=\boldsymbol{d}_t$ comes from both $\boldsymbol{o}_t$ and $\boldsymbol{v}_i$.

$$\mathbf{d}\log\boldsymbol{b}_t = \boldsymbol{q}_t\odot\underbrace{\sum_{i=1}^{t}\langle\mathbf{d}\boldsymbol{o}_t,\frac{\boldsymbol{d}_t}{\boldsymbol{d}_i}\boldsymbol{v}_i\rangle\odot\boldsymbol{b}_t\odot\boldsymbol{k}_i/\boldsymbol{b}_i}_{\mathbf{d}\boldsymbol{q}_t} - \boldsymbol{k}_t\odot\underbrace{\sum_{i=t}^{L}\langle\mathbf{d}\boldsymbol{o}_i,\frac{\boldsymbol{d}_i}{\boldsymbol{d}_t}\boldsymbol{v}_t\rangle\boldsymbol{q}_i\odot\boldsymbol{b}_i/\boldsymbol{b}_t}_{\mathbf{d}\boldsymbol{k}_t}$$

$$\mathbf{d}\log\boldsymbol{d}_t = \mathbf{d}\boldsymbol{o}_t\odot\underbrace{\sum_{i=1}^{t}\left((\boldsymbol{q}_t\odot\boldsymbol{b}_t)\left(\frac{\boldsymbol{k}_i}{\boldsymbol{b}_i}\right)^{\top}\left(\frac{\boldsymbol{v}_i}{\boldsymbol{d}_i}\right)\right)\odot\boldsymbol{d}_t}_{\boldsymbol{o}_t} - \boldsymbol{v}_t\odot\underbrace{\sum_{i=t}^{L}\left((\boldsymbol{q}_i\odot\boldsymbol{b}_i)\left(\frac{\boldsymbol{k}_t}{\boldsymbol{b}_t}\right)^{\top}\left(\frac{1}{\boldsymbol{d}_t}\right)\right)\odot\boldsymbol{d}_i}_{\mathbf{d}\boldsymbol{v}_t}$$

The trick applied there is that $\frac{\partial f(\boldsymbol{a}\odot\boldsymbol{b})}{\partial\log\boldsymbol{b}} = \boldsymbol{a}\odot\frac{\partial f(\boldsymbol{a}\odot\boldsymbol{b})}{\partial\boldsymbol{a}}$ and $\frac{\partial f(\boldsymbol{a}/\boldsymbol{b})}{\partial\log\boldsymbol{b}} = -\frac{\partial f(\boldsymbol{a}/\boldsymbol{b})}{\partial\boldsymbol{a}}\odot\boldsymbol{a}$.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_norm ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_norm ↑	COPA acc ↑	OBQA acc_norm ↑	SciQA acc ↑	BoolQ acc ↑	Avg.
0-shot													
Transformer++ 340M	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	66.0	28.4	73.8	60.9	47.7
RetNet 350M	32.33	49.19	28.6	63.5	33.5	52.5	44.5	23.4	63	28.4	73.1	60.0	47.1
Mamba 350M	28.39	39.66	30.6	65.0	35.4	50.1	46.3	23.6	71.0	28.4	73.7	52.6	47.7
GLA-Transformer 340M	28.65	43.35	30.3	64.8	34.5	51.4	45.1	22.7	70.0	29.2	73.2	58.7	48.0
0-shot													
Transformer++ 1.3B	16.85	13.44	48.9	70.8	49.6	53.6	56.0	26.5	75.0	29.8	83.6	52.3	54.6
RetNet 1.3B	18.64	17.27	43.3	70.0	47.3	52.5	54.8	25.6	70.0	31.4	82.3	57.1	53.4
Mamba 1.3B	17.06	13.89	46.2	72.2	40.1	54.1	59.0	28.2	74.0	33.0	83.1	59.1	54.9
GLA-Transformer 1.3B	17.22	14.47	46.9	71.8	49.8	53.9	57.2	26.6	73.0	32.4	84.7	58.5	55.5
5-shot													
Transformer++ 1.3B	-	16.80	42.9	70.2	50.3	53.8	60.5	28.7	75.0	33.8	90.7	46.0	55.2
RetNet 1.3B	-	23.27	37.3	69.8	47.5	51.1	58.5	27.4	72.0	31.8	87.5	45.3	52.8
Mamba 1.3B	-	23.00	31.4	71.4	51.2	54.1	60.1	30.4	79.0	33.8	88.5	47.7	55.4
GLA-Transformer 1.3B	-	18.87	41.1	71.9	49.9	54.4	61.8	28.4	75.0	34.2	90.4	56.9	56.4

Table 5: Extended zero- and five-shot performance results. All models are trained on the same subset of SlimPajama dataset with Mistral tokenizer. The 340M/1.3B models are trained for 15B/100B tokens respectively. The last column shows the average of all accuracies.

D Additional Experimental Results

The complete results on all 11 tasks, including the 5-shot results for the 1.3B models, are shown in Table 5.