Parallelizing Linear Transformers with the Delta Rule over Sequence Length

Songlin Yang[◇] **Bailin Wang**[◇] **Yu Zhang**[†] **Yikang Shen**[‡] **Yoon Kim**[◇]
[◇]Massachusetts Institute of Technology [†]Soochow University [‡]MIT-IBM Watson AI Lab
yangsl66@mit.edu

Abstract

Transformers with linear attention (i.e., linear transformers) and state-space models have recently been suggested as a viable linear-time alternative to transformers with softmax attention. However, these models still underperform transformers especially on tasks that require in-context retrieval. While more expressive variants of linear transformers which replace the additive outer-product update in linear transformers with the delta rule [DeltaNet; 85] have been found to be more effective at associative recall, existing algorithms for training such models do not parallelize over sequence length and are thus inefficient to train on modern hardware. This work describes a hardware-efficient algorithm for training linear transformers with the delta rule, which exploits a memory-efficient representation for computing products of Householder matrices [9]. This algorithm allows us to scale up DeltaNet to standard language modeling settings. We train a 1.3B model for 100B tokens and find that it outperforms recent linear-time baselines such as Mamba [25] and GLA [99] in terms of perplexity and zero-shot performance on downstream tasks (including on tasks that focus on recall). We also experiment with two hybrid models which combine DeltaNet layers with (1) sliding-window attention layers every other layer or (2) two global attention layers, and find that these hybrid models outperform strong transformer baselines.

1 Introduction

The attention mechanism [6, 95] has been shown to be an important primitive for accurate sequence modeling. Attention is moreover efficient during training as it is rich in matrix multiplications and can thus take advantage of highly parallel processing capabilities and specialized accelerators on modern GPUs. However, the complexity of attention is quadratic in sequence length, and hence it is a fundamentally expensive primitive. And while recent techniques have made it possible to scale attention to longer sequences through hardware-aware restructuring of the intermediate computations [17, 15, 49, 12], these methods still require storing the key/value vectors of previous elements, and this "KV cache" can be unwieldy to manage for long sequences.

Linear attention transformers [39] replace the exponential kernel in softmax attention with a dot-product over (possibly transformed) key and query vectors. This makes it possible to formulate linear attention as a linear RNN with matrix-valued hidden states, thus obviating the need for a KV cache and enabling constant-memory inference. While initial variants of linear attention generally underperformed softmax attention on language modeling, gated variants of linear attention which incorporate a data-dependent gating factor have recently been shown to be competitive against strong transformer baselines [99, 78, 7, 67]. These gated linear transformers, along with time-varying state space models such as Mamba [25] (which can be reparameterized as a gated linear transformer), have

The parallel DeltaNet layer is made available as part of the FLASHLINEARATTENTION library [99, 98]: https://github.com/sustcsonglin/flash-linear-attention

Preprint.

been suggested as a potential alternative to ordinary transformers. However, despite the competitive language modeling performance, these models have been shown to underperform transformers on recall-intensive tasks [5], which is important for many practical downstream tasks of interest (e.g., in retrieval-augmented generation [45]).

To enhance associative recall over long contexts, Schlag et al. [85] propose DeltaNet, a variant of a linear transformer which uses a delta rule-like update [97] to retrieve and update a value vector that is associated with the current key. DeltaNet was found to be effective on synthetic tasks and small scale language modeling/machine translation. However, the original work used a sequential algorithm that did not parallelize across sequence length, thus resulting in hardware-inefficient training, and it has not been clear how to scale DeltaNet to larger models and datasets.

This work describes a hardware-efficient training algorithm for DeltaNets which parallelizes the forward/backward passes across sequence length. We reparameterize the DeltaNet as a matrix-valued RNN whose recurrence is given by a generalized Householder transformation. This reparameterization enables the use of the memory-efficient WY representation [9] for products of Householder matrices, eliminating the need to materialize the matrix-sized hidden states (which would incur high I/O cost). The memory-efficient representation makes it possible to straightforwardly extend the chunkwise parallel strategy for training linear attention models [29, 90, 99] to the DeltaNet case. We scale DeltaNets to moderate-scale language modeling benchmarks (1.3B models trained on 100B tokens), where DeltaNet is found to obtain better language modeling and zero-shot downstream task performance than strong linear recurrent models such as Mamba [25] and GLA [99]. For in-context retrieval and learning evaluation, we evaluate DeltaNet on synthetic benchmarks such as MQAR [3], RegBench [2], MAD [71], as well as real data benchmarks tested in Arora et al. [5], where it is again found to perform well against linear recurrent baselines. Finally, we experiment with a hybrid approach where we combine DeltaNet layers with sliding attention layers or global attention layers, and find that these hybrid models can improve upon ordinary transformers, as well as the pure DeltaNet transformer.

2 Background

2.1 Linear Transformer: Transformers with Linear Attention

Given a sequence of d-dimensional input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$, transformers use the softmax attention mechanism to attend over the entire past,

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

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d}, \boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{o}_t \in \mathbb{R}^d$. (Here we assume a single attention head). Linear attention [39] replaces the exponential kernel $\exp(\boldsymbol{k}_i^\intercal \boldsymbol{q}_t)$ with the dot-product $\phi(\boldsymbol{k}_i)^\intercal\phi(\boldsymbol{q}_t)$ where $\phi : \mathbb{R}^d \to \mathbb{R}^n$ is a feature map. This makes it possible to rearrange computations to represent linear attention as a linear RNN with matrix-valued hidden states,

$$\boldsymbol{o}_t = \sum_{i=1}^{t} \frac{\phi(\boldsymbol{k}_i)^\intercal\phi(\boldsymbol{q}_t)}{\sum_{j=1}^{t}\phi(\boldsymbol{k}_j)^\intercal\phi(\boldsymbol{q}_t)}\boldsymbol{v}_i = \frac{\left(\sum_{i=1}^{t}\boldsymbol{v}_i\phi(\boldsymbol{k}_i)^\intercal\right)\phi(\boldsymbol{q}_t)}{\left(\sum_{j=1}^{t}\phi(\boldsymbol{k}_j)^\intercal\right)\phi(\boldsymbol{q}_t)} = \frac{\mathbf{S}_t\phi(\boldsymbol{q}_t)}{\mathbf{z}_t^\intercal\phi(\boldsymbol{q}_t)},$$

where $\mathbf{S}_t = \sum_{i=1}^{t}\boldsymbol{v}_i\phi(\boldsymbol{k}_i)^\intercal \in \mathbb{R}^{d\times n}$ and $\mathbf{z}_t = \sum_{i=1}^{t}\phi(\boldsymbol{k}_i) \in \mathbb{R}^n$. If we allow n to go to infinity, linear attention can use feature maps associated with polynomial kernels to compute a polynomial approximation to the exponential kernel as a dot product, and can thus approximate softmax attention arbitrarily well [5]. The denominator $\mathbf{z}_t^\intercal\phi(\boldsymbol{q}_t) \in \mathbb{R}$ can result in numerical instabilities [73] and is removed in many recent works [84, 53, 90, 99]. It is also common to use the identity mapping for ϕ [53, 90], which results in the following simplified linear transformer: $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\intercal$, $\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$.

Efficient training. Let $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times d}$ be the stacked query, key, value vectors, e.g., $\mathbf{Q}_i = \boldsymbol{q}_i$. We can then compute the output $\mathbf{O} \in \mathbb{R}^{L\times d}$ in parallel via $\mathbf{O} = \left(\mathbf{Q}\mathbf{K}^\intercal \odot \mathbf{M}_L\right)\mathbf{V}$, where $\mathbf{M}_L \in \mathbb{R}^{L\times L}$ is the causal mask. This fully "parallel form" and the above "recurrent form" have different FLOPs and memory cost tradeoffs. The parallel form takes $O(L^2d + Ld^2)$ and thus requires more FLOPs than the recurrent form, which takes $O(Ld^2)$. However, the parallel form can often be faster in practice for moderate-length sequences as it can be fully parallelized and done in $O(1)$

steps. This sequence-level parallellism also enables high GPU occupancy. While the recurrent form requires fewer FLOPs ($O(Ld^2)$), the elementwise operations involved in recurrence have low arithmetic intensity, unlike the matmul operations in the parallel form which can make use of specialized compute units (e.g., tensor cores). Thus the recurrent form is often slower in practice.[1]

Chunkwise parallel form. The chunkwise parallel form [29, 90] strikes a balance between the parallel and recurrent forms, allowing for fewer FLOPs than the parallel form and more sequence-level parallelism than the recurrent form. Concretely, suppose the query/key/value vectors are split into $\frac{L}{C}$ chunks where each chunk is of length C. Let $\mathbf{Q}_{[t]} \in \mathbb{R}^{Cd}$ be all the query vectors for chunk t, and let $\boldsymbol{q}^i_{[t]} = \boldsymbol{q}_{tC+i}$ be the i-th query vector within the t'th chunk; the key/value chunks are defined similarly. Note that $t \in [0, L/C)$, $i \in [1, C]$. The state matrices are also re-indexed such that $\mathbf{S}^i_{[t]} = \mathbf{S}_{tC+i}$, and we additionally define $\mathbf{S}^0_{[t]} = \mathbf{S}^C_{[t-1]}$, i.e., the initial state of a chunk is the last state of the previous chunk. We can then obtain the following identity for the hidden state and output vector for the r-th element within the t-th chunk,

$$\mathbf{S}^r_{[t]} = \mathbf{S}^0_{[t]} + \sum_{i=1}^{r} \boldsymbol{v}^i_{[t]} \boldsymbol{k}^{i\intercal}_{[t]}, \qquad \boldsymbol{o}^r_{[t]} = \mathbf{S}^0_{[t]} \boldsymbol{q}^r_{[t]} + \sum_{i=1}^{r} \boldsymbol{v}^i_{[t]} \left(\boldsymbol{k}^{i\intercal}_{[t]} \boldsymbol{q}^r_{[t]} \right).$$

By further rewriting the intra-chunk computation based on the parallel form, we obtain following "chunkwise parallel form",

$$\mathbf{S}_{[t+1]} = \mathbf{S}_{[t]} + \mathbf{V}^\intercal_{[t+1]} \mathbf{K}_{[t+1]} \qquad \in \mathbb{R}^{d \times d}, \tag{1}$$

$$\mathbf{O}_{[t+1]} = \mathbf{Q}_{[t+1]} \mathbf{S}^\intercal_{[t]} + \left(\mathbf{Q}_{[t+1]} \mathbf{K}^\intercal_{[t+1]} \odot \mathbf{M}_C \right) \mathbf{V}_{[t+1]} \qquad \in \mathbb{R}^{C \times d}, \tag{2}$$

where we let $\mathbf{S}_{[t]} = \mathbf{S}^0_{[t]}$ to reduce notational clutter. With this form, information is propagated chunk-to-chunk through $\mathbf{S}_{[t]}$, and the intra-chunk states $\mathbf{S}^i_{[t]}$ for $i \in [1, C]$ need not be materialized, thus saving memory.

The complexity of the chunkwise parallel form is $O(LCd + Ld^2)$, and the number of steps (without parallel scan) is $O(\frac{L}{C})$. Hence, $C = L$ recovers the fully parallel form and $C = 1$ recovers the recurrent form. The chunkwise parallel form allows us to interpolate between the two forms, in essence trading off the number of sequential computations against sequence-level parallelism. In practice C is set to a small constant (usually 64 or 128), allowing for subquadratic training. This chunkwise form enables practical speed-ups against parallel-form-only softmax attention even on moderate-length sequences, as demonstrated by FLASHLINEARATTENTION [99, 98]

2.2 DeltaNet: Linear Transformers with the Delta Update Rule

The above linear transformer employs a simple linear recurrence: $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal$. This can be seen as additively updating the memory $\mathbf{S}_{t-1}$ with new key-value associations at each time step. However, a purely additive update rule makes it difficult to deallocate past key-value associations, eventually leading to key "collisions" when $L > d$, as pointed out by Schlag et al. [84]. A model should ideally learn to remove less important key-value associations to make room for new ones, and this removal should depend on the interaction between the new key and the memory content.

DeltaNet uses the delta update rule [97] to operationalize this mechanism. Specifically, it first retrieves the old value using the current key, $\boldsymbol{v}_t^{\text{old}} = \mathbf{S}_{t-1} \boldsymbol{k}_t$. It then obtains a new value $\boldsymbol{v}_t^{\text{new}}$ by interpolating between the old value and the current value $\boldsymbol{v}_t$,

$$\boldsymbol{v}_t^{\text{new}} = \beta_t \boldsymbol{v}_t + (1 - \beta_t)\, \boldsymbol{v}_t^{\text{old}},$$

where $\beta_t = \sigma(\mathbf{W}_\beta \boldsymbol{x}_t) \in (0, 1)$. This new value is added to the memory, resulting in the following update:

$$\mathbf{S}_t = \mathbf{S}_{t-1} \underbrace{- \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^\intercal}_{\text{remove}} \underbrace{+ \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^\intercal}_{\text{write}}.$$

[1] It is possible in theory to use parallel scan [11] to parallelize the recurrent form, which would enable the computations to be performed in $O(\log L)$ steps and $O(Ld^2)$ FLOPs. However, this approach requires materializing the 2D hidden state for each time step, which would incur significant memory I/O cost unless the hidden state size is small enough that the materialization can happen in faster memory (which is the strategy adopted by Mamba [25]).

Here β_t is a "writing strength": when $\beta_t = 1$, the old value is completely removed and $\boldsymbol{v}_t^{\text{new}} = \boldsymbol{v}_t$; when $\beta_t = 0$, the memory remains unmodified and we have $\mathbf{S}_t = \mathbf{S}_{t-1}$. The output computation is the same as vanilla linear attention, i.e., $\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$. The complexity of this recurrent form is the same as that of vanilla linear attention, i.e., $O(Ld^2)$. This DeltaNet is a special case of *fast weight programmers* [86], and Schlag et al. [84] and Irie et al. [32] show that this type of linear transformer outperforms ordinary linear transformers on small-scale language modeling and synthetic retrieval tasks.

3 Parallelizing DeltaNet Across Sequence Length

In the same spirit as the chunkwise form of linear attention, we derive a chunkwise form for DeltaNet that enables hardware-efficient training through parallelizing across sequence length.

3.1 A Memory-efficient Reparameterization

We first observe that $\mathbf{S}_t$ admits a purely additive representation of the form $\mathbf{S}_t = \sum_{i=1}^{t} \boldsymbol{u}_i \boldsymbol{k}_i^\top$ for $\boldsymbol{u}_i, \boldsymbol{k}_i \in \mathbb{R}^d$, since we can simply set $\boldsymbol{u}_i = \boldsymbol{v}_i^{\text{new}} - \boldsymbol{v}_i^{\text{old}} = \beta_i(\boldsymbol{v}_i - \boldsymbol{v}_i^{\text{old}})$. Recall from §2.1 that simple linear attention has the form $\mathbf{S}_t = \sum_{i=1}^{t} \boldsymbol{v}_i \boldsymbol{k}_i^\top$. Thus, DeltaNet simply replaces the value vector $\boldsymbol{v}_i$ in linear attention with the "pseudo" value vector $\boldsymbol{u}_i$. Once the $\boldsymbol{u}_i$'s have been constructed, the rest of computation can proceed as in ordinary linear attention, i.e., $\mathbf{O} = \left(\mathbf{Q}\mathbf{K}^\top \odot \mathbf{M}\right)\mathbf{U}$ where $\mathbf{U} \in \mathbb{R}^{L \times d}$ is the row-wise concatenation of the $\boldsymbol{u}_i$ vectors.

However, computing $\boldsymbol{u}_t$ naïvely requires explicitly materializing $\mathbf{S}_{t-1}$ to compute $\boldsymbol{v}_t^{\text{old}}$, which would require $O(d^2)$ memory. We now show that we can obtain the $\boldsymbol{u}_t$'s *without* explicitly materializing $\mathbf{S}_{t-1}$ in $O(d)$ memory. Our simple proof (by induction) relies on an application of the WY representation for products of Householder matrices [9]. The base case is clear since we have $\mathbf{S}_1 = \beta_1 \boldsymbol{v}_1 \boldsymbol{k}_1^\top$, so $\boldsymbol{u}_1 = \beta_1 \boldsymbol{v}_1$. For the inductive step, we first observe that the DeltaNet update is given by,

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \boldsymbol{v}_t^{\text{old}} \boldsymbol{k}_t^\top + \boldsymbol{v}_t^{\text{new}} \boldsymbol{k}_t^\top = \mathbf{S}_{t-1} - \beta_t \left(\mathbf{S}_{t-1}\boldsymbol{k}_t\right) \boldsymbol{k}_t^\top + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top,$$

which can be seen as applying a generalized Householder transformation (i.e., matmul with an identity plus rank-one matrix) to the previous state. The inductive step is then given by,

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top = \sum_{i=1}^{t-1} \boldsymbol{u}_i \boldsymbol{k}_i^\top + \underbrace{\left(\beta_t \boldsymbol{v}_t - \beta_t \sum_{i=1}^{t-1} \boldsymbol{u}_i \left(\boldsymbol{k}_i^\top \boldsymbol{k}_t\right)\right)}_{\boldsymbol{u}_t} \boldsymbol{k}_t^\top = \sum_{i=1}^{t} \boldsymbol{u}_i \boldsymbol{k}_i^\top.$$

Note that $\boldsymbol{u}_t$ does not require materializing any of the hidden states and requires $O(d)$ memory to compute, thus completing the proof. While we have successfully avoided materializing the $\mathbf{S}_t$'s, computing the $\boldsymbol{u}_t$'s for all L (i.e., $\mathbf{U}$) takes $O(L^2 d)$ and moreover cannot be fully parallelized, unlike in linear attention where we can calculate all the value vectors $\mathbf{V}$ in parallel in $O(1)$ steps. We thus seek more a efficient chunkwise strategy for parallelizing DeltaNet, which similarly exploits memory-efficient representations of products of Householder matrices.

3.2 Chunkwise Parallel Form for DeltaNet

To derive the chunkwise parallel form, we first unroll the recurrence,

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top = \sum_{i=1}^{t} \beta_i (\boldsymbol{v}_i \boldsymbol{k}_i^\top) \left(\prod_{j=i+1}^{t} (\mathbf{I} - \beta_j \boldsymbol{k}_j \boldsymbol{k}_j^\top)\right).$$

We then define the following variables: $\mathbf{P}_i^j = \prod_{t=i}^{j}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) \in \mathbb{R}^{d \times d}$, $\mathbf{H}_i^j = \sum_{t=i}^{j} \beta_t (\boldsymbol{v}_t \boldsymbol{k}_t^\top) \mathbf{P}_{t+1}^j \in \mathbb{R}^{d \times d}$, where we let $\mathbf{P}_i^j = \mathbf{I}$ whenever $i > j$. Intuitively, $\mathbf{P}_i^j$ is the "decay factor" to be applied to $\mathbf{S}_i$ for obtaining $\mathbf{S}_j$, and $\mathbf{H}_i^j$ represents the contributions to $\mathbf{S}_j$ starting from token i. (Hence $\mathbf{S}_t = \mathbf{H}_1^t$). The chunkwise recurrence can then be written as,

$$\mathbf{S}_{[t]}^r = \mathbf{S}_{[t]}^0 \mathbf{P}_{[t]}^r + \mathbf{H}_{[t]}^r \tag{3}$$

where we define the chunkwise variables $\mathbf{S}_{[t]}^i = \mathbf{S}_{tC+i}$, $\mathbf{P}_{[t]}^r = \mathbf{P}_{tC+1}^{tC+r}$, $\mathbf{H}_{[t]}^r = \mathbf{H}_{tC+1}^{tC+r}$. Here we have $\frac{L}{C}$ chunks of size C. The trick is to now efficiently represent the $\mathbf{P}_{[t]}^r, \mathbf{H}_{[t]}^r \in \mathbb{R}^{d \times d}$ matrices

using a similar approach described in §3.1, so that these matrices can be stored in $O(d)$ memory. This is given by,

$$\mathbf{P}_{[t]}^r = \mathbf{I} - \sum_{i=1}^{r} \boldsymbol{w}_{[t]}^i \boldsymbol{k}_{[t]}^{i\intercal}, \qquad \mathbf{H}_{[t]}^r = \sum_{t=1}^{r} \boldsymbol{u}_{[t]}^i \boldsymbol{k}_{[t]}^{i\intercal} \qquad \in \mathbb{R}^{d\times d} \tag{4}$$

$$\boldsymbol{w}_{[t]}^r = \beta_{[t]}^r \left(\boldsymbol{k}_{[t]}^r - \sum_{i=1}^{r-1} \left(\boldsymbol{w}_{[t]}^i (\boldsymbol{k}_{[t]}^{i\intercal} \boldsymbol{k}_{[t]}^r) \right) \right) \qquad \in \mathbb{R}^d \tag{5}$$

$$\boldsymbol{u}_{[t]}^r = \beta_{[t]}^r \left(\boldsymbol{v}_{[t]}^r - \sum_{i=1}^{r-1} \left(\boldsymbol{u}_{[t]}^i (\boldsymbol{k}_{[t]}^{i\intercal} \boldsymbol{k}_{[t]}^r) \right) \right) \qquad \in \mathbb{R}^d \tag{6}$$

The derivations for the above can be found in the appendix. Subsequently, based on Eq. 3, we can obtain the chunk-level recurrence for hidden states and outputs as,

$$\mathbf{S}_{[t]}^r = \mathbf{S}_{[t]}^0 - \left(\mathbf{S}_{[t]}^0 \sum_{i=1}^{r} \boldsymbol{w}_{[t]}^i \boldsymbol{k}_{[t]}^{i\intercal} \right) + \sum_{i=1}^{r} \boldsymbol{u}_{[t]}^i \boldsymbol{k}_{[t]}^{i\intercal} = \mathbf{S}_{[t]}^0 + \sum_{i=1}^{r} \left(\boldsymbol{u}_{[t]}^i - \mathbf{S}_{[t]}^0 \boldsymbol{w}_{[t]}^i \right) \boldsymbol{k}_{[t]}^{i\intercal},$$

$$\boldsymbol{o}_{[t]}^r = \mathbf{S}_{[t]}^r \boldsymbol{q}_{[t]}^r = \mathbf{S}_{[t]}^0 \boldsymbol{q}_{[t]}^r + \sum_{i=1}^{r} \left(\boldsymbol{u}_{[t]}^i - \mathbf{S}_{[t]}^0 \boldsymbol{w}_{[t]}^i \right) \left(\boldsymbol{k}_{[t]}^{i\intercal} \boldsymbol{q}_{[t]}^i \right).$$

Letting $\mathbf{S}_{[t]} = \mathbf{S}_{[t]}^0$, the above can be simplified to matrix notations similarly to Eq.1-2,

$$\mathbf{S}_{[t+1]} = \mathbf{S}_{[t]} + \left(\mathbf{U}_{[t+1]} - \mathbf{W}_{[t+1]} \mathbf{S}_{[t]}^\intercal \right)^\intercal \mathbf{K}_{[t+1]} \tag{7}$$

$$\mathbf{O}_{[t+1]} = \mathbf{Q}_{[t+1]} \mathbf{S}_{[t]}^\intercal + (\mathbf{Q}_{[t+1]} \mathbf{K}_{[t+1]}^\intercal \odot \mathbf{M}) \left(\mathbf{U}_{[t+1]} - \mathbf{W}_{[t+1]} \mathbf{S}_{[t]}^\intercal \right) \tag{8}$$

where $\square_{[t]} = \square_{[t]}^{1:C} \in \mathbb{R}^{C\times d}$ for $\square \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O}, \mathbf{U}, \mathbf{W}\}$ defines the chunkwise matrices that are formed from stacking the $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{o}_t, \boldsymbol{u}_t, \boldsymbol{w}_t$ vectors.

Training speed. The sequential computations needed for constructing the chunkwise matrices $\mathbf{U}_{[t]}$'s and $\mathbf{W}_{[t]}$'s takes $O(Ld)$ work but can be parallelized across chunks, reducing the number of sequential steps from L to C. We further leverage the compact UT transform (see for details),

Once these are obtained, the chunkwise hidden state updates (Eq. 7) and the output computations (Eq. 8) are largely the same as in linear attention, and thus take $O(LCd + Ld^2)$ work and $O(\frac{L}{C})$ sequential steps.

In order to save even more memory, we discard the chunk-level hidden states $\mathbf{S}_{[t]}$'s (which would take $O(\frac{L}{C}d^2)$ space) after the forward pass and recompute them during the backward pass; the $\mathbf{U}$ and $\mathbf{W}$ matrices, which take up $O(Ld)$ space, are kept in memory.

We implement both the pure recurrent form and the chunkwise parallel form in Triton [92] and show the speed-ups for various sequence lengths (L) and head dimensions (d_{head}) in Table 1, where the model dimension d is 2048.[2] Note that our recurrent form kernel[3] is already 2x faster than the CUDA recurrent form kernel provided in the original DeltaNet paper.[4] Our chunkwise algorithm enjoys greater speed-ups as the length L and head dimension d_{head} increases.

L	d_{head}	Speed-up
2048	64	5.5x
4096	64	7.6x
8192	64	11.5x
2048	128	8.9x
4096	128	13.2x
2048	256	13.7x

Table 1: Speed comparison between the recurrent and chunkwise algorithms for DeltaNet on a single H100 GPU.

[2] So far we have been assuming a single head ($d_{\text{head}} = d$) for easier exposition. In practice we use multiple heads where the head dimension d_{head} is smaller than the model dimension d. In this case we have $\mathbf{S}_t \in \mathbb{R}^{d\times d_{\text{head}}}$.

[3] Our recurrent form kernel as well as the kernel of the original DeltaNet paper does not use parallel scan, as this would be impractical. See footnote 1.

[4] https://github.com/IDSIA/recurrent-fwp/blob/master/algorithmic/fast_weight/fast_weight_cuda.cu.

3.3 DeltaNet Transformer

We describe how the DeltaNet layer primitive is used to build up a transformer-like model using standard modules. We largely follow the LLaMA-architecture transformer [Transformer++, 94] and just replace the self-attention layer with the DeltaNet layer. We also apply normalization before output projection to stable training [73, 58]. As the additional parameters for computing scalar β_t terms are negligible, parameter allocation is roughly the same as in Transformer++: $4d^2$ for the DeltaNet layer and $8d^2$ for the SwiGLU FFN layer [87].

Feature map and normalization. Schlag et al. [84] originally follow Katharopoulos et al. [39] and apply a "ELU + 1" [14] to nonlineary transform the key/query vectors. We instead use the SiLU activation [20], which was found to perform better (as reported by Qin et al. [75]). Schlag et al. [84] also normalize the key/query vectors with the L_1 norm. We use the L_2 norm instead. Normalization has two effects: ignoring β_t, both L_1- and L_2-normalization would ensure that $\mathbf{I} - \boldsymbol{k}_t\boldsymbol{k}_t^\top$ has eigenvalues ≤ 1, making training stable; moreover, $\mathbf{I} - \boldsymbol{k}_t\boldsymbol{k}_t^\top$ is a projection matrix with L_2-normalization, and only erases information in one subspace while keeping the other $d-1$ subspace intact, which is beneficial for retaining information while still enabling some forgetting. Thus our key/query vectors are given by $\boldsymbol{k}_t = \frac{\mathrm{SiLU}(\mathbf{W}_K\boldsymbol{x}_t)}{\|\mathrm{SiLU}(\mathbf{W}_K\boldsymbol{x}_t)\|_2}$, $\boldsymbol{q}_t = \frac{\mathrm{SiLU}(\mathbf{W}_Q\boldsymbol{x}_t)}{\|\mathrm{SiLU}(\mathbf{W}_Q\boldsymbol{x}_t)\|_2}$. We ablate on these choices in our empirical study.

3.4 Hybrid Models

Following recent work on combining subquadratic token-mixing layers with existing neural network primitives [5, 18, 46], we also experiment with hybridizing DeltaNet models.

Convolutional layers. Recent linear recurrent models typically incorporate a lightweight depthwise-separable convolution layer after the query/key/value projections [25, 7, 16]. This "short convolution" layer [70] generalizes the shift SSM [22], and is efficient in both number of parameters and computational cost. We also add a short convolution layer after the query/key/value projections.

Local sliding window and global attention. Linear attention largely uses a content-based addressing mechanism [24] and lacks positional information [103]. Arora et al. [5] also argue that linear attention lacks the ability to perform precise local token shifts and comparisons, thus facing difficulties on retrieval-intensive tasks. Motivated by this, we experiment with two different hybrid architectures that incorporate softmax attention. We first explore *local sliding window attention* which attends over a sliding window and morever has been shown to significantly improve linear attention [73, 5, 48, 62]; we follow Griffin [18] and interleave DeltaNet layers and sliding multi-query local attention (SMQA) layers. We also experiment with *global attention*, which has been found to be helpful [43, 30] even if only few of the recurrent layers are replaced with full global attention [46]; we follow Fu et al. [22] to replace only two layers with global attention: the second layer and the $(\frac{N}{2}+1)$-th layer, where N is total number of layers.

4 Empirical Study

We compare the DeltaNet against strong baselines in both synthetic and real-world language modeling settings. Our main baselines include: LLaMA-architecture Transformer++ [94]; RetNet [90], a linear attention Transformer with non-data-dependent exponential decay and large head dimension; GLA [99], a linear attention Transformer with data-dependent decay; and Mamba [25], a selective state-space model with data-dependent decay. The number of parameters is kept (roughly) the same between all models.

4.1 Synthetic Benchmarks

We evaluate on three synthetic benchmarks: Multi-query associative recall [MQAR; 3], in-context language learning [RegBench; 2], and Mechanistic Architecture Design [MAD; 71].

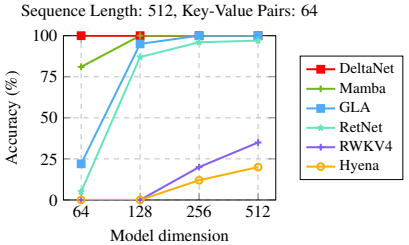

Figure 1: Accuracy (%) on MQAR.

MQAR evaluates language models' ability to (in-context) recall information within a context when faced with multiple recall queries. Arora et al. [3] show that the accuracy of MQAR has a strong correlation with the language modeling performance. We use Arora et al. [3]'s training setting and for DeltaNet we use 2 heads. We

Model	Compress	Fuzzy Recall	In-Context Recall	Memorize	Noisy Recall	Selective Copy	Average
Transformer	51.6	29.8	94.1	85.2	86.8	99.6	74.5
Hyena [70]	45.2	7.9	81.7	89.5	78.8	93.1	66.0
Multihead Hyena [55]	44.8	14.4	99.0	89.4	98.6	93.0	73.2
Mamba [25]	52.7	6.7	90.4	89.5	90.1	86.3	69.3
GLA [99]	38.8	6.9	80.8	63.3	81.6	88.6	60.0
DeltaNet	42.2	35.7	100	52.8	100	100	71.8

Table 2: Results on MAD. Results other than DeltaNet are directly borrowed from Poli et al. [71]. (Multihead) Hyena, DeltaNet and Mamba make use of convolutions, whereas GLA does not.

do not use convolutions for these experiments. Table 1 shows that DeltaNet performs perfectly (even without convolution) in the hardest setting of MQAR [3], outperforming Mamba (which uses convolution) in low dimension settings.

We next consider RegBench [2], which is a synthetic dataset designed to assess in-context learning capability of different model architectures. Each input sequence in this benchmark consists of 10 to 20 strings drawn from a probabilistic finite automaton (PFA), where each example is separated by a special separator token. Each sequence is designed to be sampled from a distinct language defined by the PFA so that a model needs to infer the underlying language from the context on the fly. During testing, a model is evaluated on predicting the next token of testing sequences generated from held-out PFAs. We follow Akyürek et al. [2] and perform a hyperparameter sweep for each model architecture and use the best result. Here again we find that DeltaNet performs strongly compared to baselines.

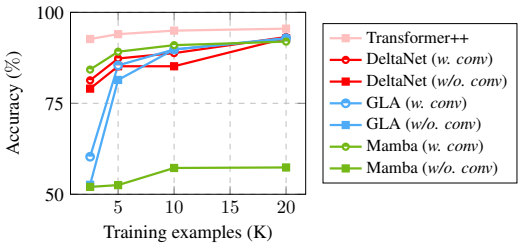

Figure 2: Accuracy (%) on RegBench.

Finally, we consider the MAD benchmark [71], a suite of synthetic token manipulation tasks designed to probe capabilities of model architectures. The results are shown in Table 2. Compared with other architectures, including MHA, DeltaNet is better at recalling tasks, especially on Fuzzy Recall, although it struggles on the "Memorize" task.

4.2 Language Modeling

Experimental setup. Following prior work [25, 99], we evaluate on Wikitext perplexity and zero-shot common sense reasoning tasks, including LAMBADA [LMB.; 64], PiQA [10], HellaSwag [Hella.; 101], WinoGrande [Wino.; 83], ARC-easy (ARC-e) and ARC-challenge (Arc-c) [13]. Following Arora et al. [5], we also evaluate the models real-world recall-intensive tasks, including FDA [4], SWDE [50], and SQUAD [80]. Both SWDE and FDA focus on extracting structured information: SWDE from raw HTML to identify semi-structured relationships, and FDA from PDFs to retrieve key-value pairs. SQUAD evaluates language models on reading comprehension by providing a text passage and a related question.

Hyperparameters. We train all models from scratch in two configurations: 340M and 1.3B parameters. Each model uses AdamW for optimization, with a peak learning rate of 3e-4. The 340M models are trained using 15 billion tokens and a batch size of 0.5M tokens, while the 1.3B models are trained with 100 billion tokens and a batch size of 2M tokens. We use a cosine learning rate schedule, starting with a warm-up phase of 0.5 billion tokens for the 340M models and 1 billion tokens for the 1.3B models. Both configurations have initial and final learning rates set at 3e-5. We apply a weight decay of 0.01 and use gradient clipping at a maximum of 1.0. The head dimension of DeltaNet is set to 128, and the kernel size for convolution layers is set at 4.

Results. Our main language modeling results are shown in Table 3. In the original works, Mamba uses convolutions by default, while GLA does not. For fair comparison, we retrain the GLA with convolution, and also train DeltaNet without convolution. For the 1.3B setting we only train the DeltaNet with convolution due to limited compute resources.

In general we find that DeltaNet outperforms the strong Mamba/GLA baselines in terms of both perplexity and downstream task performance. Improvements upon Mamba are particularly pronounced for the recall intensive tasks (i.e., SWDE, SQUAD, FDA). We also confirm the benefits of hybrid architectures [18, 46]. Both the sliding window attention hybrid and the global attention

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	Avg.	SWDE cont. ↑	SQUAD cont. ↑	FDA cont. ↑
340M params / 15B tokens												
Transformer++	28.39	42.69	31.0	63.3	34.0	50.4	44.5	24.2	41.2	42.2	22.1	21.4
RetNet (*w/o. conv*)	32.33	49.19	28.6	63.5	33.5	52.5	44.5	23.4	41.0	13.3	27.6	2.9
Mamba (*w. conv*)	28.39	39.66	30.6	65.0	35.4	50.1	46.3	23.6	41.8	12.4	23.0	2.1
GLA (*w/o. conv*)	28.65	43.35	30.3	64.8	34.5	51.4	45.1	22.7	41.5	18.6	27.2	8.1
(*w. conv*)	29.47	45.53	31.3	65.1	33.8	51.6	44.4	24.6	41.8	24.0	24.7	7.3
DeltaNet (*w/o. conv*)	29.08	50.87	30.0	63.6	33.6	51.7	46.0	23.0	41.3	24.6	26.9	4.5
DeltaNet (*w. conv*)	28.24	37.37	32.1	64.8	34.3	52.2	45.8	23.5	42.1	26.4	28.9	12.8
+ Sliding Attn	27.06	38.17	33.4	64.0	35.3	50.9	45.9	23.2	42.1	39.3	32.5	18.8
+ Global Attn (2 layers)	27.51	35.04	33.5	64.0	34.5	51.7	46.0	23.3	42.1	42.9	32.1	23.1
1.3B params / 100B tokens												
Transformer++	16.85	13.44	48.9	70.8	49.6	53.6	56.0	26.5	50.9	66.6	31.5	27.4
RetNet (*w/o. conv*)	18.64	17.27	43.3	70.0	47.3	52.5	54.8	25.6	48.9	42.8	34.7	14.3
Mamba (*w. conv*)	17.06	13.89	46.2	72.2	40.1	54.1	59.0	28.2	50.0	41.4	35.2	6.2
GLA (*w/o. conv*)	17.22	14.47	46.9	71.8	49.8	53.9	57.2	26.6	51.0	50.6	42.6	19.9
(*w. conv*)	17.25	14.92	46.2	70.6	49.9	53.0	55.3	27.0	50.4	52.4	37.4	22.3
DeltaNet (*w. conv*)	16.87	12.21	48.9	71.2	50.2	53.6	57.2	28.3	51.6	49.5	37.4	17.2
+ Sliding Attn	16.56	11.74	49.2	71.8	51.1	52.8	58.9	28.8	52.1	53.3	43.3	22.3
+ Global Attn (2 layers)	16.55	12.40	48.8	70.8	50.7	54.2	58.4	28.1	51.8	71.0	43.0	29.8
DeltaNet Ablations (340M)												
w. L_1-norm & 1+ELU	31.12	55.96	26.3	63.9	33.0	50.9	44.3	21.8	40.1	14.5	23.9	6.2
w. L_2-norm & 1+ELU	28.03	37.62	32.2	65.7	34.7	51.8	45.4	22.5	42.1	23.8	28.6	13.1
w. L_2-norm & ReLU	28.75	43.53	30.2	64.0	33.9	48.9	45.6	22.8	40.9	27.2	26.7	9.0

Table 3: Main language modeling results against Transformer++, RetNet [90], Mamba [25], and GLA [99]. All models are trained on the same subset of the SlimPajama dataset with the Mistral tokenizer. The Transformer++, RetNet, Mamba, GLA (*w/o. conv*) results are taking from Yang et al. [99]. For hybrid models, "Sliding Attn" interleaves a sliding window attention every other layer, and "Global Attn" uses full global attention on two layers. The 340M/1.3B models are trained for 15B/100B tokens respectively. All results are obtained through `lm-evaluation-harness` [23].

hybrid work well, outperforming the Transformer++ in terms of perplexity. The hybrid DeltaNet that just replaces two DeltaNet layers with global attention outperforms a Transformer++ even on recall-intensive benchmarks.

Ablation study. In Table 3 (bottom) we ablate the choice of feature map and normalization. The original DeltaNet uses L_1-norm and the ELU feature map. We find that simply replacing the L_1-norm with the L_2-norm greatly increases performance. As for feature map, we experiment with $\{\texttt{ReLU}, 1 + \texttt{ELU}, \texttt{SiLU}\}$ and find that `SiLU` performs the best.

Training throughput We compare the training throughputs of different 1.3B models in different training lengths and batch size settings. The result is shown in Figure 3. We observe that see that the training speed of DeltaNet is close to GLA and significantly faster than Mamba. All linear-time models outperform Transformer++ in longer-sequence training settings.

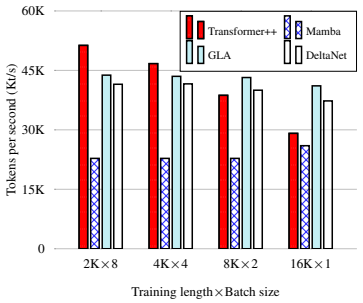

Figure 3: Training throughput of 1.3B models on a single H100.

5 Discussion

5.1 DeltaNet vs. State Space Models / Linear RNNs

To discuss DeltaNet against existing linear RNNs (including state-space models) we first discuss a general class of associative RNNs with matrix-valued hidden states. Given a matrix-valued hidden state $\mathbf{S}_t \in \mathbb{R}^{d \times n}$ and current input $\boldsymbol{x}_t \in \mathbb{R}^d$, we are interested in sequence models with the following form:

$$\mathbf{S}_t = \mathbf{S}_{t-1} \bullet \mathbf{M}_t + \boldsymbol{v}_t \boldsymbol{k}_t^\top, \qquad \text{(recurrence)}$$
$$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t, \qquad \text{(memory read-out)}$$

where $\bullet$ is an associative operator (e.g., Hadamard product, matrix multiplication, etc.). The matrix $\mathbf{M}_t$ and vectors $\boldsymbol{v}_t$, $\boldsymbol{k}_t$, $\boldsymbol{q}_t$ are (potentially non-linear) functions of the current input $\boldsymbol{x}_t$.

As is the case in vector-valued linear RNNs [54, 88], the use of an associative operator enables the use of parallel scan [11] to calculate $\mathbf{S}_1, \ldots, \mathbf{S}_L$ in $O(\log L)$ steps and $O(L)$ work (ignoring the terms associated with the associative operation) if the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L$ are given (though see our discussion in footnote 1). Hence, as long as the associative operator is not too expensive, training can be efficient. However, parallel scan by itself is not sufficient for training language models at

Model	Recurrence	Memory read-out
Linear Attention [39, 38]	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
+ Kernel	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\phi(\boldsymbol{k}_t)^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\phi(\boldsymbol{q}_t)$
+ Normalization	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\phi(\boldsymbol{k}_t)^\top,\ \ \boldsymbol{z}_t = \boldsymbol{z}_{t-1} + \phi(\boldsymbol{k}_t)$	$\boldsymbol{o}_t = \mathbf{S}_t\phi(\boldsymbol{q}_t)/(\boldsymbol{z}_t^\top\phi(\boldsymbol{q}_t))$
DeltaNet [84]	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^\top) + \beta_t\boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
Gated RFA [68]	$\mathbf{S}_t = g_t\mathbf{S}_{t-1} + (1-g_t)\boldsymbol{v}_t\boldsymbol{k}_t^\top,\ \ \boldsymbol{z}_t = g_t\boldsymbol{z}_{t-1} + (1-g_t)\boldsymbol{k}_t$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t/(\boldsymbol{z}_t^\top\boldsymbol{q}_t)$
S4 [26]	$\mathbf{S}_t = \mathbf{S}_{t-1}\odot\exp(-(\boldsymbol{\alpha}\mathbf{1}^\top)\odot\exp(\boldsymbol{A})) + \boldsymbol{B}\odot(\boldsymbol{v}_t\mathbf{1}^\top)$	$\boldsymbol{o}_t = (\mathbf{S}_t\odot\boldsymbol{C})\mathbf{1} + \boldsymbol{d}\odot\boldsymbol{v}_t$
DFW [53, 40]	$\mathbf{S}_t = \mathbf{S}_{t-1}\odot(\boldsymbol{\beta}_t\boldsymbol{\alpha}_t^\top) + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
RetNet [90, 75]	$\mathbf{S}_t = \gamma\mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
Mamba [25]	$\mathbf{S}_t = \mathbf{S}_{t-1}\odot\exp(-(\boldsymbol{\alpha}_t\mathbf{1}^\top)\odot\exp(\boldsymbol{A})) + (\boldsymbol{\alpha}_t\odot\boldsymbol{v}_t)\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t + \boldsymbol{d}\odot\boldsymbol{v}_t$
GLA [99]	$\mathbf{S}_t = \mathbf{S}_{t-1}\odot(\mathbf{1}\boldsymbol{\alpha}_t^\top) + \boldsymbol{v}_t\boldsymbol{k}_t^\top = \mathbf{S}_{t-1}\mathrm{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
RWKV-6 [67]	$\mathbf{S}_t = \mathbf{S}_{t-1}\mathrm{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = (\mathbf{S}_{t-1} + (\boldsymbol{d}\odot\boldsymbol{v}_t)\boldsymbol{k}_t^\top)\boldsymbol{q}_t$
HGRN-2 [78]	$\mathbf{S}_t = \mathbf{S}_{t-1}\mathrm{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t(\mathbf{1}-\boldsymbol{\alpha}_t)^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$
mLSTM [7]	$\mathbf{S}_t = f_t\mathbf{S}_{t-1} + i_t\boldsymbol{v}_t\boldsymbol{k}_t^\top,\ \ \boldsymbol{z}_t = f_t\boldsymbol{z}_{t-1} + i_t\boldsymbol{k}_t$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t/\max\{1, \lvert\boldsymbol{z}_t^\top\boldsymbol{q}_t\rvert\}$
Mamba-2 [16]	$\mathbf{S}_t = \gamma_t\mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t\boldsymbol{q}_t$

Table 4: Overview of recent linear recurrent models that have been proposed and applied to autoregressive language modeling (ordered in rough chronological order). These works make use of a matrix-valued hidden state $\mathbf{S}_t \in \mathbb{R}^{d\times n}$ that is updated through an associative recurrence followed by an outer-product-based addition. Here $\odot$ is the Hadamard product. Some models make use of an additional linear RNN with hidden state vector $\boldsymbol{z}_t$, which used to normalized the query vector $\boldsymbol{q}_t$. Variables with the subscript t (e.g., $\boldsymbol{v}_t, \boldsymbol{\alpha}_t, f_t, \gamma_t$) are (potentially non-linear) functions of the current input $\boldsymbol{x}_t$. Non-time-varying parameters (e.g., $\boldsymbol{A}, \boldsymbol{d}, \gamma$) are denoted without subscripts; these parameters are either learned or set to fixed values. Matrices are denoted with bold upper case letters, vectors with bold lower case, and scalars with italic letters. Many models make use of a kernel ϕ (e.g., [84, 68]) but we subsume them into the key/value vectors to reduce notational clutter. S4 denoted here uses the diagonal transition transition [27, 88] with the same discretization as in Mamba.

practical scale due to some associative operator's being too expensive. Recent models such as such as Mamba [25] and gated linear attention Transformers [90, 99, 78, 67, 7] thus make use of cheap element-wise recurrence updates, in particular the Hadamard product, i.e., $\bullet = \odot$. See Table 4 for how recent models can be cast into this form.

Standard matrix multiplications (i.e., $\mathbf{S}_{t-1}\bullet\mathbf{M}_t = \mathbf{S}_{t-1}\mathbf{M}_t$) on the other hand can model richer interactions that go beyond elementwise recurrence. Without any structural assumptions on $\mathbf{M}_t$ however, these operations would take $O(dn^2)$ for each update (as opposed to $O(dn)$ for elementwise products), which would be prohibitively expensive. Hence, DeltaNet's use of $\mathbf{M}_t = \mathbf{I} - \beta_t\boldsymbol{k}_t\boldsymbol{k}_t^\top$ can be seen as exploiting structured matrices to efficiently model interactions beyond elementwise recurrences. Note that our algorithm generalizes to a more general class of matrices of the form $\mathbf{M}_t = \mathbf{I} - \boldsymbol{a}_t\boldsymbol{b}_t^\top$. We adopt the DeltaNet parameterization in the present work as we are primarily interested in improving recall (through DeltaNet's key-value update rule) while maintaining parameter-efficiency; we leave the exploration of alternative parameterizations to future work.

5.2 Towards a Unifying Framework for Efficient Autoregressive Sequence Transformations

While the above class of models (i.e., matrix-valued RNNs with associative recurrence and outer-product-based additive) makes it possible to unify recent models, we do not claim that it is the "right" level at which view (autoregressive) sequence transformations of the form $\{\boldsymbol{x}_t\}_{t=1}^L \mapsto \{\boldsymbol{o}_t\}_{t=1}^L$, where $\boldsymbol{o}_t$ cannot depend on any $\boldsymbol{x}_j$ if $j > t$. For example, this framing makes it difficult to (neatly) capture other subquadratic models that have been shown to be effective [100, 41, 82, 69, 70]. An alternative unifying framework might be to view the above sequence transformations as a discretization of a continuous state space model [26, 88, 25], or as a matrix multiplication with a masked structured matrix [63, 74, 37, 16]. What does seem important, however, is that a framework should ideally expose efficient algorithms for training, and the algorithm should be hardware-efficient, which, in the case of modern GPUs, means that it should be rich in (half-precision) matrix multiplications. From this perspective, the state-space duality (SSD) framework recently proposed by Dao and Gu [16], which provides a connection between SSM-based sequence transformations and structured matrix multiplications with a semiseparable matrix, seems a promising candidate. For example, SSD provides a formal perspective on the "chunkwise parallel algorithm" for training gated linear attention transformers [90, 99] (which was originally proposed by Hua et al. [29] to train a hybrid transformer). However, this framework may not capture an important class of models, e.g., models where the associative recurrence involves matrix multiplication with an unstructured matrix, or models that make use of more exotic associative operators.

Finally, we observe that there have been many recent works that have been proposed which purportedly match or outperform classic transformers while maintaining subquadratic training and inference. As can be seen in Table 4, the "token-mixing" component of these works are closely related to one another. However, the way in which the token-mixing primitive is used to build up a transformer-like model varies widely. For example, while most recent works make use of depthwise-separable convolution layers (not shown in Table 4) [25, 66, 79, 7, 16], earlier works generally do not [39, 85, 68]. As we show in our RegBench (Figure 2) and language modeling (Table 3) experiments, the use of convolution layers is crucial for some models. There are also differences in the parameterizations of the feedforward layers used for the "channel mixing" component. Such variations should be taken into account before declaring a particular model layer superior to another.

5.3 Limitations

Our work has several limitations. First, in terms of computation, although we propose a new hardware-efficient algorithm, the training speed still lags behind that of GLA. This is due to the overhead caused by modeling state-to-state dependencies as described above, which requires "marginalizing" over the head dimension inside the kernel, similar to the case of softmax attention. However, for GLA since there are intra-state dependencies (everything is elementwise), and thus it is easy to use tiling to support arbitrary size of head dimension, as implemented in Yang and Zhang [98]. This limitation would potentially limit DeltaNet's memory size.

Second, we found that the length generalization of DeltaNet was limited, while GLA and RetNet (and Mamba to an extent) have been found to be able to extrapolate beyond the training length [99]. We speculate that this is because DeltaNet lacks explicit decay factors. This could be improved through incorporating a decay term in the recurrence, e.g., $\mathbf{M}_t = \gamma_t(\mathbf{I} - \boldsymbol{k}_t\boldsymbol{k}_t^\top)$ for $\gamma_t \in (0, 1)$, although the decay mechanism could potentially hurt in-context retrieval.[5]

6 Related Work

Chunkwise linear attention. Hua et al. [29] first proposed chunkwise form for linear attention; however, they used a hybrid linear and nonlinear attention model similar to Munkhdalai et al. [61]. It is possible to adapt their algorithm to compute the *exact* output of the pure linear attention, as shown in Sun et al. [90] and Yang et al. [99]. The chunkwise linear attention algorithm has also been independently discovered in several works [90, 36, 16]. Yang et al. [99] and Qin et al. [77] discuss I/O-aware hardware optimization for chunkwise linear attention and Sun et al. [89] make generalization to multi-node distributed training. Inspired by the chunkwise form, we propose a new algorithm for hardware-efficient DeltaNet training, significantly improving the training efficiency and allowing for large-scale experiments.

Delta rule and memory capacity. Linear transformers can be seen as a type of iterated Hopfield networks [60], and this connection can provide perspectives on the limitations and improvements of linear attention transformers. For example, vanilla linear transformers use a Hebbian-like update rule, which has been shown to have limited memory capacity [57]. Later works in Hopfield networks use higher-order polynomials [19] and exponential kernels [81, 42] to enhance the memory capacity, which is also related to attention with polynomial kernels explored in PolysketchFormer [36] and Based Linear Attention [5, 1]. On the other hand, the delta rule has been shown to have better memory capacity [72, 47]. In this sense, given the fixed size recurrent state, using the delta rule is able to achieve a better frontier of the recall-memory tradeoff curve [5]. Delta rule has demonstrated better performance compared to the additive rule used in vanilla linear Transformer [84, 32, 33, 31, 34]. Recently, Munkhdalai et al. [61] also show that delta rule-enhanced linear attention is better in retrieval and summarization tasks.

Householder matrices with WY representation. Householder matrices, known for preserving norms, are a type of orthogonal matrix extensively used in machine learning [56, 59, 102, 93, 76, 8]. These matrices allow for efficient computation of inverses and their Jacobian determinant of one, making them particularly suitable for applications in normalizing flows [56, 8]. Notably, Mathiasen et al. [56] developed a chunkwise fast algorithm for computing the cumulative product of Householder matrices for normalizing flows, leveraging the WY representation. Our approach, while sharing the same high-level concept, tackles a different problem and is arguably more general.

[5] However we found the DeltaNet + local sliding-window attention hybrid to generalize well, which could provide an appealing middle ground.

There has also been significant interest in using orthogonal matrices to parameterize the transition matrices of RNNs [59, 35, 96, 28] for mitigating vanishing gradients. Mhammedi et al. [59] use the WY representation to reduce the memory footprint when training nonlinear RNNs with Householder transition matrices.

Hybrid models. There has been much recent work on developing hybrid mdoels by comining linear recurrent layers (state-space models, linear recurrent Transformers, linear RNNs) with local chunk attention [51, 104, 21, 52, 61] or sliding window attention [104, 5, 18] or global attention [43, 44, 30, 22, 46, 65, 91]. Poli et al. [71] systematically study the scaling law of hybrid models. We similarly show that combining DeltaNet with classic attention is an effective strategy.

7 Conclusion

We describe an algorithm that parallelizes DeltaNet training across the sequence length dimension, achieving significant speed-ups against existing implementations. This makes it possible to scale up DeltaNet to moderate-scale language modeling settings, where we find that it performs well compared to recent linear-recurrent baselines. In addition to the algorithmic contribution, we also experiment with hybridizing DeltaNet layers with short convolution, sliding window attention, and global attention layers, and find that such hybrid models perform well.

Acknowledgements

This study was supported by funds from MIT-IBM Watson AI Lab. We thank Simran Arora and Liliang Ren for helpful discussion. We thank Michael Poli and Armin Thomas for sharing the raw results from the MAD benchmark experiment. We thank Kazuki Irie for useful feeback on the paper draft.

References

[1] Y. Aksenov, N. Balagansky, S. M. L. C. Vaina, B. Shaposhnikov, A. Gorbatovski, and D. Gavrilov. Linear Transformers with Learnable Kernel Functions are Better In-Context Models, June 2024. URL http://arxiv.org/abs/2402.10644. arXiv:2402.10644 [cs].

[2] E. Akyürek, B. Wang, Y. Kim, and J. Andreas. In-Context Language Learning: Architectures and Algorithms, Jan. 2024. URL http://arxiv.org/abs/2401.12973. arXiv:2401.12973 [cs].

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

[4] S. Arora, B. Yang, S. Eyuboglu, A. Narayan, A. Hojel, I. Trummer, and C. Ré. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes, Apr. 2023. arXiv:2304.09433 [cs].

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

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[8] R. v. d. Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. Sylvester Normalizing Flows for Variational Inference, Feb. 2019. URL http://arxiv.org/abs/1803.05649. arXiv:1803.05649 [cs, stat].

[9] C. H. Bischof and C. V. Loan. The WY representation for products of householder matrices. In *SIAM Conference on Parallel Processing for Scientific Computing*, 1985. URL https://api.semanticscholar.org/CorpusID:36094006.

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

[11] G. E. Blelloch. Prefix sums and their applications. 1990.

[12] W. Brandon, A. Nrusimha, K. Qian, Z. Ankner, T. Jin, Z. Song, and J. Ragan-Kelley. Striped Attention: Faster Ring Attention for Causal Transformers. *ArXiv*, abs/2311.09431, 2023.

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

[14] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), Feb. 2016. URL http://arxiv.org/abs/1511.07289. arXiv:1511.07289 [cs].

[15] T. Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *CoRR*, abs/2307.08691, 2023. doi: 10.48550/ARXIV.2307.08691. arXiv: 2307.08691.

[16] T. Dao and A. Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv: 2405.21060*, 2024.

[17] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *NeurIPS*, 2022.

[18] S. De, S. L. Smith, A. Fernando, A. Botev, G. Cristian-Muraru, A. Gu, R. Haroun, L. Berrada, Y. Chen, S. Srinivasan, G. Desjardins, A. Doucet, D. Budden, Y. W. Teh, R. Pascanu, N. De Freitas, and C. Gulcehre. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models, Feb. 2024. URL http://arxiv.org/abs/2402.19427. arXiv:2402.19427 [cs].

[19] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168(2):288–299, July 2017. ISSN 0022-4715, 1572-9613. doi: 10.1007/s10955-017-1806-y. URL http://arxiv.org/abs/1702.01929. arXiv:1702.01929 [math].

[20] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning, Nov. 2017. URL http://arxiv.org/abs/1702.03118. arXiv:1702.03118 [cs].

[21] M. Fathi, J. Pilault, P.-L. Bacon, C. Pal, O. Firat, and R. Goroshin. Block-state transformer. *arXiv preprint arXiv:2306.09539*, 2023.

[22] D. Y. Fu, T. Dao, K. K. Saab, A. W. Thomas, A. Rudra, and C. Ré. 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*, 2023.

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

[24] A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines, Dec. 2014. URL http://arxiv.org/abs/1410.5401. arXiv:1410.5401 [cs].

[25] A. Gu and T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. 2023.

[26] A. Gu, K. Goel, and C. Ré. Efficiently Modeling Long Sequences with Structured State Spaces, 2022. 2111.00396.

[27] A. Gupta, A. Gu, and J. Berant. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

[28] K. Helfrich, D. Willmott, and Q. Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978. PMLR, 2018.

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

[30] F. Huang, K. Lu, C. Yuxi, Z. Qin, Y. Fang, G. Tian, and G. Li. Encoding recurrence into transformers. In *The Eleventh International Conference on Learning Representations*, 2022.

[31] K. Irie and J. Schmidhuber. Images as weight matrices: Sequential image generation through synaptic learning rules. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=ddad0PNUvV.

[32] K. Irie, I. Schlag, R. Csord'as, and J. Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *ArXiv*, abs/2106.06295, 2021. URL https://api.semanticscholar.org/CorpusID:235417174.

[33] K. Irie, F. Faccio, and J. Schmidhuber. Neural differential equations for learning to program neural nets through continuous learning rules. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/fc09b26b85ab3abb2832bd555a2e4215-Abstract-Conference.html.

[34] K. Irie, R. Csordás, and J. Schmidhuber. Practical computational power of linear transformers and their recurrent and self-referential extensions. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9455–9465, Singapore, Dec. 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.588. URL https://aclanthology.org/2023.emnlp-main.588.

[35] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio. Gated Orthogonal Recurrent Units: On Learning to Forget. *Neural Computation*, 31(4):765–783, Apr. 2019. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco_a_01174. URL https://direct.mit.edu/neco/article/31/4/765-783/8458.

[36] P. Kacham, V. Mirrokni, and P. Zhong. Polysketchformer: Fast transformers via sketches for polynomial kernels. *arXiv preprint arXiv:2310.01655*, 2023.

[37] Y. Kang, G. Tran, and H. De Sterck. Fast multipole attention: A divide-and-conquer attention mechanism for long sequences. *arXiv preprint arXiv:2310.11960*, 2023.

[38] J. Kasai, H. Peng, Y. Zhang, D. Yogatama, G. Ilharco, N. Pappas, Y. Mao, W. Chen, and N. A. Smith. Finetuning Pretrained Transformers into RNNs. In M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 10630–10643. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.830.

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

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

[41] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[42] D. Krotov and J. Hopfield. Large Associative Memory Problem in Neurobiology and Machine Learning, Apr. 2021. URL http://arxiv.org/abs/2008.06996. arXiv:2008.06996 [cond-mat, q-bio, stat].

[43] T. Lei. When Attention Meets Fast Recurrence: Training Language Models with Reduced Compute. In M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7633–7648, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.602. URL https://aclanthology.org/2021.emnlp-main.602.

[44] T. Lei, R. Tian, J. Bastings, and A. P. Parikh. Simple recurrence improves masked language models. *arXiv preprint arXiv:2205.11588*, 2022.

[45] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Apr. 2021. URL http://arxiv.org/abs/2005.11401. arXiv:2005.11401 [cs].

[46] O. Lieber, B. Lenz, H. Bata, G. Cohen, J. Osin, I. Dalmedigos, E. Safahi, S. Meirom, Y. Belinkov, S. Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[47] K. C. Lingashetty. Delta learning rule for the active sites model. *arXiv preprint arXiv:1007.0417*, 2010.

[48] L. D. Lingle. Transformer-vq: Linear-time transformers via vector quantization. *arXiv preprint arXiv:2309.16354*, 2023.

[49] H. Liu, M. Zaharia, and P. Abbeel. Ring Attention with Blockwise Transformers for Near-Infinite Context. *ArXiv*, abs/2310.01889, 2023.

[50] C. Lockard, P. Shiralkar, and X. L. Dong. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. In J. Burstein, C. Doran, and T. Solorio, editors, *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)*, pages 3047–3056, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1309. URL https://aclanthology.org/N19-1309.

[51] X. Ma, C. Zhou, X. Kong, J. He, L. Gui, G. Neubig, J. May, and L. Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

[52] X. Ma, X. Yang, W. Xiong, B. Chen, L. Yu, H. Zhang, J. May, L. Zettlemoyer, O. Levy, and C. Zhou. Megalodon: Efficient llm pretraining and inference with unlimited context length. *arXiv preprint arXiv:2404.08801*, 2024.

[53] H. H. Mao. Fine-Tuning Pre-trained Transformers into Decaying Fast Weights. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10236–10242, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.697.

[54] E. Martin and C. Cundy. 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.

[55] S. Massaroli, M. Poli, D. Y. Fu, H. Kumbong, R. N. Parnichkun, A. Timalsina, D. W. Romero, Q. McIntyre, B. Chen, A. Rudra, C. Zhang, C. Ré, S. Ermon, and Y. Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions. *ArXiv*, abs/2310.18780, 2023. URL https://api.semanticscholar.org/CorpusID:264590326.

[56] A. Mathiasen, F. Hvilshøj, J. R. Jørgensen, A. Nasery, and D. Mottin. Faster orthogonal parameterization with householder matrices. In *ICML, Workshop Proceedings*, 2020.

[57] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the hopfield associative memory. *IEEE Trans. Inf. Theory*, 33:461–482, 1987.

[58] J. Mercat, I. Vasiljevic, S. Keh, K. Arora, A. Dave, A. Gaidon, and T. Kollar. Linearizing large language models. *arXiv preprint arXiv:2405.06640*, 2024.

[59] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections, June 2017. URL http://arxiv.org/abs/1612.00188. arXiv:1612.00188 [cs].

[60] B. Millidge. Linear Attention as Iterated Hopfield Networks. URL http://www.beren.io/2024-03-03-Linear-Attention-as-Iterated-Hopfield-Networks/.

[61] T. Munkhdalai, M. Faruqui, and S. Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.

[62] Y. Nahshan, J. Kampeas, and E. Haleva. Linear Log-Normal Attention with Unbiased Concentration, Feb. 2024. URL http://arxiv.org/abs/2311.13541. arXiv:2311.13541 [cs].

[63] T. Nguyen, V. Suliafu, S. Osher, L. Chen, and B. Wang. Fmmformer: Efficient and flexible transformer via decomposed near-field and far-field attention. *Advances in neural information processing systems*, 34:29449–29463, 2021.

[64] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context, June 2016. URL http://arxiv.org/abs/1606.06031. arXiv:1606.06031 [cs].

[65] J. Park, J. Park, Z. Xiong, N. Lee, J. Cho, S. Oymak, K. Lee, and D. Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks. *arXiv preprint arXiv:2402.04248*, 2024.

[66] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. G. V, X. He, H. Hou, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, K. S. I. Mantri, F. Mom, A. Saito, X. Tang, B. Wang, J. S. Wind, S. Wozniak, R. Zhang, Z. Zhang, Q. Zhao, P. Zhou, J. Zhu, and R.-J. Zhu. RWKV: Reinventing RNNs for the Transformer Era. *CoRR*, abs/2305.13048, 2023. doi: 10.48550/ARXIV.2305.13048. arXiv: 2305.13048.

[67] B. Peng, D. Goldstein, Q. Anthony, A. Albalak, E. Alcaide, S. Biderman, E. Cheah, X. Du, T. Ferdinan, H. Hou, P. Kazienko, K. K. GV, J. Kocoń, B. Koptyra, S. Krishna, R. McClelland Jr., N. Muennighoff, F. Obeid, A. Saito, G. Song, H. Tu, S. Woźniak, R. Zhang, B. Zhao, Q. Zhao, P. Zhou, J. Zhu, and R.-J. Zhu. Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence, Apr. 2024. URL http://arxiv.org/abs/2404.05892. arXiv:2404.05892 [cs].

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

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

[70] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, S. Baccus, Y. Bengio, S. Ermon, and C. Ré. Hyena Hierarchy: Towards Larger Convolutional Language Models. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 28043–28078. PMLR, 2023.

[71] M. Poli, A. W. Thomas, E. Nguyen, P. Ponnusamy, B. Deiseroth, K. Kersting, T. Suzuki, B. Hie, S. Ermon, C. Ré, C. Zhang, and S. Massaroli. Mechanistic Design and Scaling of Hybrid Architectures, Mar. 2024. arXiv:2403.17844 [cs].

[72] D. Prados and S. Kak. Neural network capacity using delta rule. *Electronics Letters*, 3(25): 197–199, 1989.

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

[74] Z. Qin, X. Han, W. Sun, B. He, D. Li, D. Li, Y. Dai, L. Kong, and Y. Zhong. Toeplitz neural network for sequence modeling. *arXiv preprint arXiv:2305.04749*, 2023.

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

[76] Z. Qin, W. Sun, K. Lu, H. Deng, D. Li, X. Han, Y. Dai, L. Kong, and Y. Zhong. Linearized Relative Positional Encoding, July 2023. URL http://arxiv.org/abs/2307.09270. arXiv:2307.09270 [cs].

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

[78] Z. Qin, S. Yang, W. Sun, X. Shen, D. Li, W. Sun, and Y. Zhong. HGRN2: Gated Linear RNNs with State Expansion. 2024. URL https://api.semanticscholar.org/CorpusID:269043328.

[79] Z. Qin, S. Yang, and Y. Zhong. Hierarchically gated recurrent neural network for sequence modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

[80] P. Rajpurkar, R. Jia, and P. Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Melbourne, Australia, 2018. Association for Computational Linguistics.

[81] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. Hopfield Networks is All You Need, Apr. 2021. URL http://arxiv.org/abs/2008.02217. arXiv:2008.02217 [cs, stat].

[82] A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9: 53–68, 2021.

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

[84] I. Schlag, K. Irie, and J. Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9355–9366. PMLR, 2021.

[85] I. Schlag, T. Munkhdalai, and J. Schmidhuber. Learning Associative Inference Using Fast Weight Memory, Feb. 2021. URL http://arxiv.org/abs/2011.07831. arXiv:2011.07831 [cs].

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

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

[88] J. T. H. Smith, A. Warrington, and S. W. Linderman. 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.

[89] W. Sun, Z. Qin, D. Li, X. Shen, Y. Qiao, and Y. Zhong. Linear attention sequence parallelism. *arXiv preprint arXiv:2404.02882*, 2024.

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

[91] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024.

[92] P. Tillet, H. Kung, and D. D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019*, pages 10–19. ACM, 2019. doi: 10.1145/3315508.3329973.

[93] J. M. Tomczak and M. Welling. Improving Variational Auto-Encoders using Householder Flow, Jan. 2017. arXiv:1611.09630 [cs, stat].

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

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

[96] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pages 3570–3578. PMLR, 2017.

[97] B. Widrow, M. E. Hoff, et al. Adaptive switching circuits. In *IRE WESCON convention record*, volume 4, pages 96–104. New York, 1960.

[98] S. Yang and Y. Zhang. FLA: A Triton-Based Library for Hardware-Efficient Implementations of Linear Attention Mechanism, Jan. 2024. URL https://github.com/sustcsonglin/flash-linear-attention. original-date: 2023-12-20T06:50:18Z.

[99] S. Yang, B. Wang, Y. Shen, R. Panda, and Y. Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

[100] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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

[102] J. Zhang, Q. Lei, and I. S. Dhillon. Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization, Mar. 2018. arXiv:1803.09327 [cs, stat].

[103] J. Zhang, S. Jiang, J. Feng, L. Zheng, and L. Kong. Linear Attention via Orthogonal Memory, 2023. arXiv:2312.11135.

[104] Q. Zhang, D. Ram, C. Hawkins, S. Zha, and T. Zhao. Efficient long-range transformers: You need to attend more, but not necessarily at every layer. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

A WY representation

For simplicity we discuss the first chunk here.

We first show $\mathbf{P}_n = \mathbf{I} - \sum_{t=1}^{n} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal$ by induction,

$$
\begin{aligned}
\mathbf{P}_n &= \prod_{t=1}^{n} (\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\intercal) \\
&= \mathbf{P}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= (\mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal)(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal + (\sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n-1} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal - \underbrace{\left(\beta_n \boldsymbol{k}_n - \beta_n \sum_{t=1}^{n-1} \left(\boldsymbol{w}_t (\boldsymbol{k}_t^\intercal \boldsymbol{k}_n) \right) \right)}_{\boldsymbol{w}_n} \boldsymbol{k}_n^\intercal \\
&= \mathbf{I} - \sum_{t=1}^{n} \boldsymbol{w}_t \boldsymbol{k}_t^\intercal
\end{aligned}
$$

Similarly, we show $\mathbf{S}_n = \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^\intercal$ by induction,

$$
\begin{aligned}
\mathbf{S}_n &= \mathbf{S}_{n-1}(\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal \right) (\mathbf{I} - \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal) + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal - \left(\sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal \right) \beta_n \boldsymbol{k}_n \boldsymbol{k}_n^\intercal + \beta_n \boldsymbol{v}_n \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n-1} \boldsymbol{u}_t \boldsymbol{k}_t^\intercal + \underbrace{\left(\beta_n \boldsymbol{v}_n - \beta_n \sum_{t=1}^{n-1} \boldsymbol{u}_t \left(\boldsymbol{k}_t^\intercal \boldsymbol{k}_n \right) \right)}_{\boldsymbol{u}_n} \boldsymbol{k}_n^\intercal \\
&= \sum_{t=1}^{n} \boldsymbol{u}_t \boldsymbol{k}_n^\intercal
\end{aligned}
$$