Gated KalmaNet: A Fading Memory Layer Through Test-Time Ridge Regression

Liangzu Peng[1,*] Aditya Chattopadhyay[2,†] Luca Zancato[2] Elvis Nunez[2] Wei Xia[2] Stefano Soatto[2]
University of Pennsylvania[1] AWS Agentic AI[2]
lpenn@seas.upenn.edu {achatto,zancato,elvisnun,wxia,soattos}@amazon.com

Abstract

As efficient alternatives to softmax Attention, linear state-space models (SSMs) achieve constant memory and linear compute, but maintain only a lossy, fading summary of the past, often leading to inferior performance in recall oriented settings. We propose Gated KalmaNet (GKA), a layer that reduces this gap by accounting for the full past when predicting the next token, while maintaining SSM-style efficiency. GKA is inspired by the Kalman Filter and solves ridge regression problems online at test time, with constant memory and linear time in the sequence length. An insight is that standard Kalman filter equations are numerically unstable in low-precision environments (e.g., bfloat16) and difficult to parallelize on modern GPUs. We address both challenges via two innovations: (1) an adaptive regularization strategy with input-dependent gating that controls the condition number of the problem, ensuring numerical stability and balancing memory retention; (2) the use of Chebyshev Iteration instead of conventional iterative solvers, which we show to be more stable in low-precision settings. To improve scalability, we implement Chebyshev Iteration in a hardware-aware, chunk-wise manner, along with custom kernels for back-propagating through our adaptive regularization and gating mechanisms. On short-context tasks, GKA shows strong language understanding capabilities and outperforms existing SSMs (e.g., Mamba2, and Gated DeltaNet). On long-context tasks, GKA excels at real-world RAG and LongQA tasks up to 128k tokens with more than 10% relative improvement over baselines. Finally, we show GKA outperforms Mamba when extended for ImageNet classification.

1. Introduction

Large Language Models (LLMs) powered by (softmax) Attention mechanisms [61] have revolutionized sequence modeling through their ability to form rich associations within their context window. However, a fundamental challenge that LLMs face is that their time complexity scales quadratically and storage grows linearly with their input length.

Recent years have seen intense efforts to develop Attention alternatives. Among them, memory layers based on linear State-Space models (SSMs) have grown popular for their linear-time computation and constant storage cost in the sequence length [9, 68]. These SSMs find inspirations from classic techniques in adaptive signal processing, and integrating them into modern SSMs leads to principled layer design and enhanced performance [38, 70, 75]. However, pure SSM models still underperform Attention in many settings, especially on long-context tasks. This gap is a consequence of their different memory mechanisms: SSMs have a *fading* fixed dimensional *lossy* state of the past, while Attention has an *eidetic* ever increasing KV-cache state [75].

To bridge this gap, we aim at designing a memory layer that enjoys the efficiency of linear SSMs while performing computation conditioned on the exact past. Towards this goal, we first draw insights from the *Kalman filter* (KF) [30]. In signal processing terms, KF computes the most recent state conditioned on all data seen thus far, and, under mild assumptions, KF is optimal in the *Maximum A-Posteriori* (MAP) sense. In the LLM context, we use KF to update the state of an SSM layer and predict its output based on all past inputs. However, integrating KF into such a layer is non-trivial and faces two challenges:

- *Parallelizable Training.* KF is an online algorithm and needs to be parallelized to fully utilize modern hardware that is highly optimized for large-scale LLM training.
- *Numerical Stability.* KF involves matrix inversion, which can be numerically unstable in low precision arithmetic.

In this work, we propose *Gated KalmaNet* (GKA), a memory layer that incorporates KF into its design and is both numerically stable and trainable on highly parallelizable hardware. We start by observing that the KF recursion solves a test-time ridge regression problem. Then, to solve such a regularized problem stably, we make the following choices:

- At the modeling level, we adaptively choose the regularization strength of our test-time objective function based on the Frobenius norm of the regularized data covariance. With this choice we can easily upper bound the condition

*Work done during an internship at AWS Agentic AI.
†Correspondence to achatto@amazon.com

number of the optimization problem.

- At the algorithmic level, we note that exact solvers (e.g., `torch.linalg.solve`) are hard to parallelize (in a chunk-wise manner), so we resort to the classic *Chebyshev Iteration* (CH), which we show has high numerical accuracy and fast convergence compared with alternatives such as (accelerated) gradient descent and conjugate gradient.

To make GKA scalable and efficient, we implement CH with adaptive regularization in Triton in a hardware-aware, chunk-wise manner. Our technical novelty here includes deriving a chunk-wise implementation that back-propagates through the Frobenius norm, for which the difficulty is the presence of a *nested* recurrence. Furthermore, we combine CH with a *gating mechanism* that decides the regression residual weights in an input-aware and time-varying fashion, enhancing the contribution of recent inputs and smoothly fading out distant contexts. Overall, to the best of our knowledge, this is a first adoption of the CH method for training sequence modeling layers in LLMs stably at scale.

Finally, we demonstrate the efficacy of GKA on several benchmarks. On synthetic recall tasks (MQAR) [1], GKA achieves the highest recall among state-of-the-art linear SSMs including Mamba2 [9] and (Gated) DeltaNet [70, 71]. Also, GKA outperforms existing SSMs on several short-context tasks (from LM-Harness [16]) and long-context tasks (from RULER [27] and HELMET [72]). Specifically, GKA improves upon SSM baselines by at least 10% on real-world long-context tasks like Retrieval-Augmented Generation and Long Question-Answering tasks up to 128k tokens.

2. Prior Work and Preliminaries

In this section we briefly review prior work and preliminaries that will set the stage for motivating our work. For a more detailed exposition of related work see Section A.

(Softmax) Attention. At each time t, Attention [61] linearly projects the t-th input token to obtain three vectors, named *query* q_t, *key* k_t, *value* v_t respectively. Then, it outputs a vector $y_t \in \mathbb{R}^D$ as a convex combination of all values seen so far, with coefficients $c_1, \dots, c_t$ given by inner products of current query q_t with all seen keys and a softmax mapping:

$$y_t = \sum_{i=1}^{t} c_i v_i, \qquad c_i := \frac{\exp(\frac{k_i^\top q_t}{\sqrt{D}})}{\sum_{i=1}^{t} \exp(\frac{k_i^\top q_t}{\sqrt{D}})}. \tag{Attn}$$

From an optimization perspective, (Attn) can be viewed as solving the following *regression* objective[1],

$$y_t = \operatorname*{argmin}_{v} \sum_{i=1}^{t} \exp\left(\frac{k_i^\top q_t}{\sqrt{D}}\right) \cdot \|v - v_i\|_2^2. \tag{1}$$

[1]Concretely, for keys and queries of unit norm, Attention is precisely the *Nadaraya-Watson estimator* [43, 66] with the Gaussian kernel to approximate the conditional expectation of the value given a query; cf. [4, 62].

The success of (Attn) is often attributed to its ability to perform verbatim retrieval of relevant context from the entire past. Here, the past refers to the entire key-value pairs observed thus far, also known as the *KV-cache*, which grows linearly with time t. Moreover, the computation is also linear at each time t, and doing so for all t results in a quadratic time complexity. This high computation and storage cost of Attention makes its use prohibitive in long context scenarios.

Linear State-Space Models (SSMs). The high computation cost of (Attn) has motivated a flurry of work developing new LLM layers, like SSMs, with linear rather than quadratic cost. Most SSMs maintain a *state matrix* $S_t \in \mathbb{R}^{D \times D}$ and update it at each time step via a linear recursion of the form

$$S_t = \gamma_t \cdot S_{t-1} + \beta_t \cdot v_t k_t^\top, \quad y_t = S_t q_t, \tag{Linear-SSM}$$

where γ_t, β_t are typically in $[0, 1]$. Unlike the verbatim lookup of (Attn), here (Linear-SSM) essentially compresses the entire KV-cache into a fixed-dimensional representation S_t. Subsequent computation of the output y_t relies on S_t and no longer on the exact past. This results in a constant cost of storage and computation at every timestep.

In many linear SSMs (e.g., RetNet [59], Mamba2 [9]), the use of γ_t and β_t is often heuristic and finds inspirations from nonlinear recurrent neural networks [25]; in that light, γ_t and β_t are called *forgetting* and *input* gates, respectively. This basic form of (Linear-SSM) has been generalized by replacing γ_t with a diagonal matrix (GLA [68], RWKV-6 [48], Longhorn [38]) or *low-rank-plus-identity* matrix (Gated DeltaNet [56, 70, 71], DeltaProduct [58], RWKV-7 [49]).

Similarly to that of (Attn) the case with *low-rank-plus-identity* matrices can often be justified from an optimization perspective. For example, Gated DeltaNet [56, 70] updates the state via (I is the $D \times D$ identity matrix)

$$S_t = \gamma_t \cdot S_{t-1} \left(I - \beta_t k_t k_t^\top\right) + \beta_t \cdot v_t k_t^\top, \tag{GDN}$$

which can be viewed as applying one gradient descent step with step-size β_t and initialization $\gamma_t S_{t-1}$ to the objective

$$\min_{S} \|S k_t - v_t\|_2^2. \tag{2}$$

The objectives of (GDN) (2) and (Attn) (1) are prime examples that expose a general distinction between linear SSMs and (Attn): The former updates its state based on a regression objective that considers only the previous *lossy* state and the current time step, whereas the latter uses the entire, exact KV-cache to solve its regression objective (1).

We hypothesize this myopic view of SSM objectives results in their lower performance and limited long-context abilities. We then ask: *What is an objective or, equivalently, a recursion that considers the entire past as (Attn) while still being solvable in linear time as in (Linear-SSM)?*

3. A Linear SSM Inspired by the Kalman Filter

In Section 3.1 we show how the Kalman Filter (KF) gives insights into a new linear SSM layer that takes all past time instants into account. In Section 3.2 we explain the numerical and efficiency challenges of building such a layer.

3.1. Motivation from Kalman Filter

KF is an established online approach that takes the exact past into account to optimally solve a *weighted ridge regression* objective (e.g., see [50, Proposition 2 & Lemma 3]). In our context, this means that the *optimal* state

$$S_t = \underset{S\in\mathbb{R}^{D\times D}}{\operatorname{argmin}} \lambda \cdot \|S\|_{\mathrm{F}}^2 + \sum_{i=1}^{t} \eta_i \cdot \|Sk_i - v_i\|_2^2 \qquad (3)$$

can be computed by the KF recursion

$$S_t = S_{t-1} - \frac{(S_{t-1}k_t - v_t)k_t^\top \Phi_{t-1}}{1/\eta_t + k_t^\top \Phi_{t-1} k_t}, \qquad \text{(KF)}$$

where η_t is the weight for the t-th key-value pair, and Φ_{t-1} is the Hessian inverse of (3) at time $t-1$ (Φ_{t-1} itself can be continually updated via the *Woodbury matrix identity*). It is now clear that objective (3) takes the entire KV-cache into account, similarly to (Attn). It is also clear that (KF) is an efficient update scheme similarly to (Linear-SSM); indeed, (KF) is also a *low-rank-plus-identity* form (cf. (GDN)).

While (Linear-SSM) relies on instantaneous objectives akin to (2) (cf. [70, Table 2]), (KF) leverages second-order information from Φ_{t-1} to solve (3) optimally. It is in this sense that we say (KF) is more expressive than other (Linear-SSM) or (GDN) (see also [51, Sec. 6]). We now detail the differences in the objectives of (KF) and (Attn):

- (KF) computes a parametric linear estimator that enables a constant-sized memory, while (1) computes a non-parametric point estimate that entails storing the full cache.
- In (1), the weights of the same residual vary over time as the queries differ at each time, while in (3) i-th weight η_i is constant once observed at time i. The former results in quadratically many weights—thus a quadratic time complexity—and the latter linearly many.
- In (3), the regularizer $\lambda\cdot\|S\|_{\mathrm{F}}^2$ prevents overfitting our state to key-value pairs, as only a finite amount of "information" can be stored in a constant-sized memory beyond which will result in "fuzzy" recall. In this light, λ can be thought of as controlling the memorization "capacity" of the state.

3.2. Hurdles Towards Scalable Kalman Filter SSMs

Despite its optimality and (sequential) efficiency the (KF) recursion lacks a hardware-aware implementation that leverages parallelism in modern Tensors Cores. Moreover, for long sequences it can lose numerical precision due to division (and due to how Φ_t is updated). The final hurdle is conceptual: Fixing weights η_i and regularization λ over time as in (3) might make a layer less expressive.

We are aware of the use of (KF) or (3) in neural network training three decades ago [57] or in deep continual learning recently [40, 52, 76]. We are also aware of the recent mentioning of (3) or efforts towards solving it, which go by the name *test-time optimization* [63, 65]. However, to the best of our knowledge, none of the prior work has fully addressed the above hurdles that need to be solved to design an SSM layer that is trainable in parallel, numerical well-behaved, and sufficiently expressive. In particular, both [63] and [65] have overlooked a basic numerical concern: The worst-case numerical error in solving (3) can be $\epsilon\cdot\kappa$ [20], where κ is the condition number of the Hessian in (3) and ϵ the machine precision; since $\epsilon\approx 0.007$ (bf16), (3) has to be regularized *strongly* for κ and the worst-case error to be small, regardless of algorithmic choices to solve (3). Indeed, the regularization enforced in [63] sets λ to be lower bounded by 0.25, but this is not sufficient: Their κ is as large as 500 [63, Fig. 13], implying a worst-case error of 3.5 (The implementation of [63] available on GitHub is numerically vulnerable; we failed to train it without NaNs in various settings.). Also, the regression objective in [65] has no regularization, which makes it numerically ill-posed for low-precision training.

4. Gated KalmaNet (GKA)

We propose *Gated KalmaNet* (GKA) to address the above hurdles: We enhance numerical stability via adaptive regularization and the classic *Chebyshev Iteration* (CH), increase expressivity of KF via a standard gating mechanism, and improve parallelism via a hardware-friendly implementation.

4.1. CH with Adaptive Regularization & Weighting

Motivation. As alluded, solving (3) via (KF) is sequential in nature, and here we consider alternatives amenable to parallelizable training. Our first step towards this is to write down a closed form solution to (3) and compute the output

$$y_t = S_t q_t = \left(\sum_{i=1}^{t} \eta_i v_i k_i^\top\right)\left(\sum_{i=1}^{t} \eta_i k_i k_i^\top + \lambda I\right)^{-1} q_t.$$

With the weighted covariances $U_t := \sum_{i=1}^{t} \eta_i v_i k_i^\top$ and $H_t := \sum_{i=1}^{t} \eta_i k_i k_i^\top$, we note that y_t can be computed via first solving $(H_t+\lambda I)x = q_t$ for x and then left-multiplying U_t. An exact solver (e.g., `torch.linalg.solve`) can do so with high accuracy by parallelizing over the batch dimension. However, it is inefficient for two reasons. First, it takes $O(D^3)$ time for every t. Second, it requires explicitly forming and materializing all H_t's, which entails large I/O costs. In light of this, we resort to first-order iterative methods. These methods use matrix-vector products to enable chunk-wise parallelism over batches without materializing all H_t's. Furthermore, they often take $O(D^2)$ time

Algorithm 1: Chebyshev Iteration to solve $H\xi = q$

Input: $H \in \mathbb{R}^{D\times D}, q \in \mathbb{R}^D$, eigenvalue bounds L, μ with $L \geq \mu > 0$, number of iterations r;

Initialize: $\rho \leftarrow \frac{L-\mu}{L+\mu}$; $\omega_0 = 2$; the first two iterates $\xi_{-1} \leftarrow 0, \xi_0 \leftarrow \frac{2q}{L+\mu}$;

For Loop $(i = 1, \ldots, r)$:

$$\omega_i \leftarrow \frac{4}{4 - \rho^2\omega_{i-1}} \qquad \text{(weight schedule)}$$

$$\xi_i \leftarrow \xi_{i-1} - \frac{2\cdot\omega_i}{L+\mu}(H\xi_{i-1} - q) \qquad \text{(grad descent)}$$

$$\xi_i \leftarrow \xi_i + (\omega_i - 1)(\xi_{i-1} - \xi_{i-2}) \qquad \text{(momentum)}$$

Output: ξ_r

complexity per iteration and can converge quickly in a few iterations. The iterative method we choose is the *Chebyshev Iteration* (CH); we proceed to describe its basic idea, with the justification for using CH deferred to Section 4.2.3.

Chebyshev Iteration (CH). CH is an *accelerated gradient descent* method (AGD) that applies (grad descent) and (momentum) to the quadratic objective $\frac{1}{2}\xi^\top H\xi - \xi^\top q$, that is to solve $H\xi = q$ (Algorithm 1). Different from vanilla AGD, CH incorporates a (weight schedule) and makes specific choices of different parameters; these choices make CH optimal with the fastest convergence among first-order methods [47]. We replace the above exact solver with CH:

$$\hat{x}_t = \mathrm{CH}(H_t + \lambda I, q_t, r), \qquad y_t = U_t\hat{x}_t.$$

Here, $\mathrm{CH}(H_t + \lambda I, q_t, r)$ means r iterations of CH to approximately solve $(H_t + \lambda_t I)x = q_t$. To improve stability and expressivity, next we allow regularization λ and weight η_i to be time-varying and chosen adaptively. We write λ_t and $\eta_{t,i}$ to make their dependency in time t explicit, with $\eta_{t,i}$ being the weight of the i-th token at time t.

Adaptive Regularization. As mentioned, the condition number κ_t of $H_t + \lambda_t I$ has to be controlled for any method to be numerically stable. We choose λ_t to be proportional to the Frobenius norm $\|H_t\|_{\mathrm{F}}$, that is to set $\lambda_t = a \cdot \|H_t\|_{\mathrm{F}}$ for some constant $a > 0$. An upper bound on κ_t now ensures:

$$\kappa_t = \frac{\lambda_{\max}(H_t) + \lambda_t}{\lambda_{\min}(H_t) + \lambda_t} \leq \frac{\|H_t\|_{\mathrm{F}} + \lambda_t}{\lambda_t} = \frac{a+1}{a}. \qquad (4)$$

Here $\lambda_{\max}(H_t), \lambda_{\min}(H_t)$ are the maximum and minimum eigenvalues of H_t, respectively. Given this choice of λ_t, we set $L = \|H_t\|_{\mathrm{F}} + \lambda_t$ and $\mu = \lambda_t$ for Algorithm 1.

Adaptive Weighting (Gating). We use weights $\eta_{t,i}$ that are exponentially decaying in time: For all $t \geq i$, we parameterize $\eta_{t,i} = \prod_{j=i+1}^{t}\gamma_j$, with each $\gamma_j \in [0,1]$ learnable. The fading weights encode the "prior" of *recency bias* that has been shown to exist in LLMs [14] without even explicitly computing the weights from the query-key dot products as in (Attn). Similarly to (Attn), the weights on the residuals are now time-varying, but differently, the exponentially decay parameterization allows for linear-time implementation.

Forward Recurrence. We now summarize our recurrence which arms CH with adaptive regularization and weighting:

$$H_t = \gamma_t \cdot H_{t-1} + k_t k_t^\top, \quad U_t = \gamma_t \cdot U_{t-1} + v_t k_t^\top, \qquad y_t = U_t\hat{x}_t, \quad \hat{x}_t = \mathrm{CH}(H_t + \lambda_t I, q_t, r). \qquad \text{(CH)}$$

4.2. Chunk-wise Implementation

Here we describe our implementation for the forward + backward passes for (CH). For more details, see Section D.

4.2.1. Forward Pass

Similarly to [9, 68, 70], we now describe a *chunk-wise* implementation for (CH). In (CH), given U_t and $\hat{x}_t$, computing $y_t = U_t\hat{x}_t$ in a chunk-wise fashion is similar to that of (Linear-SSM); also similar is the calculation of $H_t\xi_{i-1}$ as needed in (grad descent). For these we refer the reader to [68, 70] for details. Our algorithmic novelty here is a chunk-wise computational formula for $\|H_t\|_{\mathrm{F}}$, presented next.

Let T be the sequence length and C the chunk size such that $N := T/C$ is an integer. For $t = 0, \ldots, N-1$, write $[t] := tC$. The core idea of a chunk-wise implementation is as follows. First, we compute and store the *initial state* $H_{[t]}$ of every chunk. This gives us implicit access to $H_{[t]+c}$ via unrolling the recurrence of H_t for c steps and therefore allows us to carry out computation with $H_{[t]+c}$; for example, we can compute the matrix-vector product $H_{[t]+1}\xi$ via $H_{[t]}\xi + \gamma_1 k_{[t]+1}k_{[t]+1}^\top\xi$. This is without forming $H_{[t]+1}$ explicitly, thereby reducing the number of states to materialize on chip. To implement such a scheme, we need to precompute all $H_{[t]}$'s sequentially, and then do the computation with parallelism over chunks and within each chunk.

We now make this idea precise for computing all $\|H_t\|_{\mathrm{F}}$'s within a chunk. Since the computation of each chunk is the same, we simplify by working with the first one where we have access to initial state H_0, gates $\gamma_1, \ldots, \gamma_C$, keys $K_C = [k_1, \ldots, k_C] \in \mathbb{R}^{D\times C}$,and we aim to compute $\|H_1\|_{\mathrm{F}}, \|H_2\|_{\mathrm{F}}, \ldots, \|H_C\|_{\mathrm{F}}$. With these notations, we first compute the C-dimensional vector $\zeta = [\zeta_1, \ldots, \zeta_C]^\top$ of cumulative products of γ_i's, with $\zeta_c = \prod_{i=1}^{c}\gamma_i$. Then, form the $C \times C$ upper triangular matrix M whose (i,j)-th entry $M_{j,c}$ is ζ_c/ζ_j $(\forall c \geq j)$. Now, unroll the recurrence of H_c:

$$H_c = \zeta_c H_0 + \sum_{j=1}^{c} M_{j,c}k_j k_j^\top = \zeta_c H_0 + \sum_{j=1}^{C} M_{j,c}k_j k_j^\top.$$

Expanding $\|H_c\|_{\mathrm{F}}^2$ gives the following sum of three terms:

$$\zeta_c^2\|H_0\|_{\mathrm{F}}^2 + 2\zeta_c\sum_{j=1}^{C} M_{j,c}k_j^\top H_0 k_j + \Big\|\sum_{j=1}^{C} M_{j,c}k_j k_j^\top\Big\|_{\mathrm{F}}^2.$$

With ζ, the first term $\zeta_c^2 \cdot \|H_0\|_{\mathrm{F}}^2$ is easily computed in parallel for all c. For the second term, we first compute the vector of quadratic forms $k_j^\top H_0 k_j$ for all j in parallel, broadcast it and multiply it with M element-wise, sum over each column, and multiply the result with 2ζ element-wise. Finally, with Gram matrix $G_C := K_C^\top K_C$, one verifies the third term can be computed in parallel for all c via the following pseudocode:

$$\text{column-sum}(((G_C \odot G_C)M) \odot M). \tag{5}$$

Here $\odot$ denotes element-wise multiplication and the sum is over each column. Summing the three terms and taking the square root, we obtain $\|H_1\|_{\mathrm{F}}, \ldots, \|H_C\|_{\mathrm{F}}$, as desired.

4.2.2. Backward Pass

Motivation. Typically, the backward pass is done automatically via `torch.autograd`. However, for iterative methods such as CH (Algorithm 1), `torch.autograd` would store some *activations* or intermediate iterates, entailing large storage cost. While in principle we can back-propagate through CH without storing any intermediate activations or iterates (by our trick of *reverting the CH iterations*, cf. Section E.1), under this trick it is difficult to compute all the gradients in a chunk-wise fashion. Therefore, we resort to the *implicit differentiation* trick, which is practically efficient and chunk-wise implementable, for backpropagation through the linear equations that CH approximately solves.

Implicit Differentiation. We derive the backward pass for our method with the standard implicit differentiation trick. It assumes we find an exact solution x_t^* to the equations $(H_t + \lambda_t I)x = q_t$. In the backward pass, we are given the gradient $dx_t^* := \frac{d\mathcal{L}}{dx_t^*}$ of some loss function $\mathcal{L}$, and need to compute the corresponding gradients at q_t, k_t, γ_t. For example, via the chain rule we obtain $dq_t := \frac{d\mathcal{L}}{dq_t}$ via

$$dq_t = (H_t + \lambda_t I)^{-1} dx_t^*, \tag{6}$$

that is to solve linear equations similarly to the forward pass. Since the forward pass computes an approximate solution $\hat{x}_t$ via CH, we receive an approximate up stream gradient $d\hat{x}_t$ (not exactly dx_t^*). Thus we employ CH to obtain an approximate gradient $d\hat{q}_t = \mathrm{CH}(H_t + \lambda_t I, d\hat{x}_t, r)$; cf. Table 1.

Backward Recurrence. Besides dq_t, we need to compute dH_t from which we obtain dk_t and $d\gamma_t$ via the chain rule. We describe $d\gamma_t$ in the Appendix. Here we analyze dk_t:

Lemma 1. *With $\lambda_t = a\|H_t\|_F$, $w_t = \frac{2a \cdot (x_t^*)^\top dq_t}{\|H_t\|_F}$, we have*

$$dk_i = \sum_{t \geq i} M_{i,t} \left(-dq_t (x_t^*)^\top - x_t^* dq_t^\top - w_t H_t \right) k_i. \tag{7}$$

With $A_i := \sum_{t \geq i} M_{i,t} \cdot dq_t (x_t^*)^\top$, we can compute the first two terms $-A_i k_i$ and $-A_i^\top k_i$ in (7), similarly to (Linear-SSM). Specifically, A_i satisfies the recursion

$$A_i = \gamma_{i+1} A_{i+1} + dq_i (x_i^*)^\top, \tag{8}$$

thus calculating $A_i k_i$ amounts to calculating $U_t q_t$ in (Linear-SSM); a difference is that the recursion here runs backwards.

Similarly, with $B_i = \sum_{t \geq i} M_{i,t} w_t H_t$, the third term in (7) can be written recursively as

$$B_i = \gamma_{i+1} B_{i+1} + w_i H_i, \quad o_i = B_i k_i. \tag{9}$$

Chunk-wise Recurrence. As indicated, a chunk-wise implementation for computing $A_i k_i$ is known. On the other hand, computing $B_i k_i$ is more challenging than $A_i k_i$, as the additive term $w_i H_i$ in the backward recursion (9) is not necessarily rank-1; rather, H_i itself is defined via the forward recursion in (CH). Our contribution here is a derivation for computing $B_i k_i$ efficiently in a chunk-wise manner.

We begin by unrolling B_i to B_{C+1}:

$$\begin{aligned} B_i =& M_{i,C} \cdot \gamma_{C+1} B_{C+1} + B_i^{\text{intra}} \\ B_i^{\text{intra}} :=& \sum_{c=i}^{C} M_{i,c} \cdot w_c H_c \end{aligned} \tag{10}$$

We next discuss the *intra-chunk* term $B_i^{\text{intra}} k_i$ and *cross-chunk* term $M_{i,C} \cdot \gamma_{C+1} B_{C+1}$ in succession.

Intra-chunk Computation. We now unroll H_c and obtain an expression B_i^{intra} more amenable to parallelism:

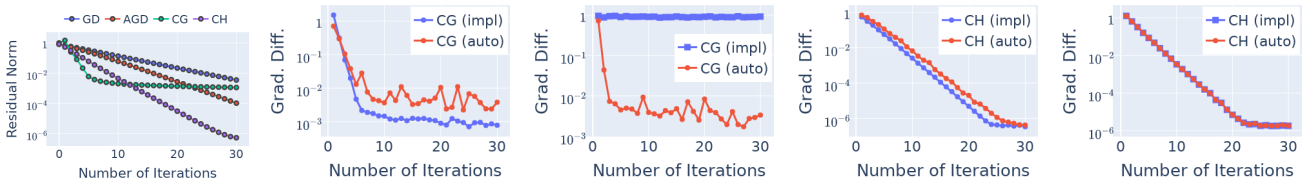
$$\begin{aligned} B_i^{\text{intra}} &= \sum_{c=i}^{C} M_{i,c} \cdot w_c H_c \\ &= \sum_{c=1}^{C} M_{i,c} w_c \Big(\zeta_c H_0 + \sum_{j=1}^{C} M_{j,c} k_j k_j^\top \Big) \\ &= H_0 \sum_{c=1}^{C} M_{i,c} w_c \zeta_c + \sum_{j=1}^{C} k_j k_j^\top \sum_{c=1}^{C} M_{i,c} w_c M_{j,c}. \end{aligned}$$

The coefficients of H_0, written as b_i, are easily computed in parallel for all i via element-wise operations, broadcasting, and summing. The coefficient of $k_j k_j^\top$ is precisely the (i,j)-th entry of the matrix $M_w := M \operatorname{diag}(w_1, \ldots, w_C) M^\top$. Thus $[B_1^{\text{intra}} k_1, \ldots, B_C^{\text{intra}} k_C]$ is equal to

$$H_0(\operatorname{diag}(b_1, \ldots, b_C) K_C) + K_C \left((K_C^\top K_C) \odot M_w \right).$$

The *mask* M_w is in general a full matrix with no zero entries, as opposed to the triangular matrix in the case of (Linear-SSM). While the triangular mask in the backward pass allows the error feedback from future tokens to be leveraged for learning past tokens, here our full mask M_w allows all tokens to interact with all other tokens in the backward pass, which facilitates the information flow and learning.

Cross-chunk Computation. In (10), both γ_{C+1} and B_{C+1} are from the future chunks, thus we revise (10) into the cross chunk recursion of $\widetilde{B}_{C+1} := \gamma_{C+1} B_{C+1}$ which allows us



(a) Empirical convergence (b) CG as a single layer (c) CG in 5-layer LLAMA (d) CH as a single layer (e) CH in 5-layer LLAMA

Figure 1. **CH converges with smaller errors than CG and is more numerically stable.** Convergence of different methods in residual norms during the forward pass with batch size 8, sequence length 2048, 8 heads, head dimension 128 (a), and relative gradient differences from the exact solver (`torch.linalg.solve`) to CG (b, c) or CH (d, e). The backward pass is via *implicit differentiation* (*impl*) or `torch.autograd` (*auto*); cf. Table 1. In (b, d) the gradients are those of $[q_t, k_t]$; in (c, e) the gradients are those of network weights.

Table 1. **Implicit differentiation for computing dq_t.**

	forward pass	backward pass
exact	$x_t^* = (H_t + \lambda_t I)^{-1} q_t$	$dq_t^* = (H_t + \lambda_t I)^{-1} dx_t^*$
CG	$\hat{x}_t = \mathrm{CG}(H_t + \lambda_t I, q_t, r)$	$d\hat{q}_t = \mathrm{CG}(H_t + \lambda_t I, d\hat{x}_t, r)$
CH	$\hat{x}_t = \mathrm{CH}(H_t + \lambda_t I, q_t, r)$	$d\hat{q}_t = \mathrm{CH}(H_t + \lambda_t I, d\hat{x}_t, r)$

to maintain a single term $\widetilde{B}_{C+1}$ from the future:

$$\widetilde{B}_1 = \zeta_C \cdot \widetilde{B}_{C+1} + \widetilde{B}_1^{\text{intra}}, \quad \widetilde{B}_1^{\text{intra}} := \sum_{c=i}^{C} \zeta_c \cdot w_c H_c.$$

In our intra-chunk computation, we store the intra-chunk term $\widetilde{B}_1^{\text{intra}}$ of all chunks, implement the above with a simple for loop, and collect the terms $\zeta_C \cdot \widetilde{B}_{C+1} k_i$.

4.2.3. Comparison to Other Iterative Solvers

Here we validate our choice of Chebyshev Iteration (CH) by benchmarking it against other iterative methods.

Convergence in the Forward Pass. We generate random regression problems, which we solve via CH and 3 other baselines: gradient descent (GD), accelerated GD with Nesterov's momentum (AGD), conjugate gradient (CG). GD and AGD are run with stepsizes that are optimal for regression problems. Fig. 1a shows CG converges the fastest within a few iterations, while CH reaches the same accuracy as CG at iteration 10 and eventually attains the smallest errors.

Stability of the Backward Pass. We then proceed and measure the gradient stability of CG and CH, whose backward passes are implemented either via implicit differentiation as per Table 1 (*impl*), or via `torch.autograd` (*auto*).

In Fig. 1b, CG (impl) as a standalone layer has its gradient close to that of the exact solver up to a 10^{-3} relative difference. In Fig. 1c, this difference is amplified to almost 1 in a 5-layer LLAMA where (Attn) is replaced with (3). This indicates CG (impl) completely deviates from the reference gradient (exact), defeating its purpose of training the network from the regression feedback. In contrast, the gradients of CH (impl) and CH (auto) are eventually close to that of the exact solver either as a single layer or within multiple layers (Fig. 1c, d), up to a 10^{-6} difference. Moreover, the curves for CH (impl) and CH (auto) nearly overlap, suggesting their gradients may be close. Lemma 2 formalizes this intuition and justifies our choice of CH over the alternatives:

Lemma 2. *Let dq_t be the exact gradient of q_t for CH, e.g., computed by CH (auto). Let $d\hat{q}_t$ be the gradient of CH (impl), computed as per Table 1. We have $dq_t = d\hat{q}_t$.*

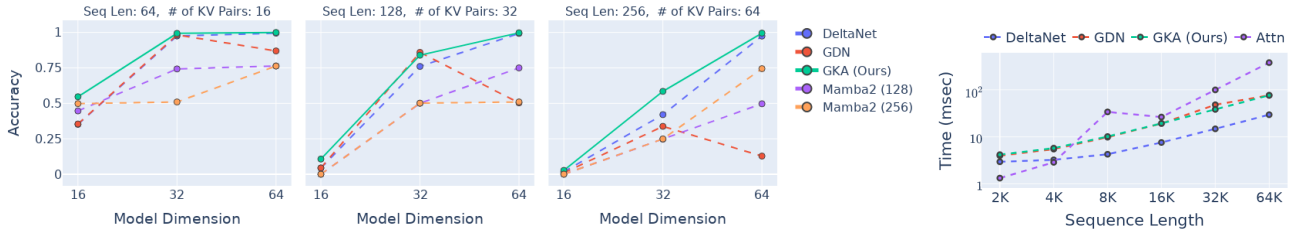
5. Experiments

In this section, we empirically validate the efficacy of GKA. We first evaluate memorization ability on synthetic associative recall tasks (Section 5.1). We then report training throughput of GKA (Section 5.2). Next, we examine performance on short-context language understanding benchmarks such as commonsense reasoning and long-context modeling abilities in Section 5.3. Finally, we demonstrate the effectiveness of GKA beyond language modeling by evaluating on ImageNet classification (Section 6). The Appendix details our experimental settings (Section F) and ablations of various modeling choices (Section J, Section K).

Baselines. All experiments consider the following state-of-the-art linear SSM-based fading memory layers as baselines: Mamba2 [9], DeltaNet [70], Gated DeltaNet (GDN) [71], and Gated Linear Attention (GLA) [68]. Each of these layers rely on instantaneous objectives that depend on the previous *lossy* state and current tokens (e.g., (2)), as opposed to the entire history of tokens observed so far as in GKA. Finally, we contrast our results with (Softmax) Attention, which serves as our paragon. For our Attention-based model, we adopt the architecture proposed in Qwen3 models [67].

5.1. GKA on Synthetic Associative Recall Tasks

We first assess the capability of our models to recall information on the multi-Query Associative Recall (MQAR) task [1]. This task presents the model with a sequence of key-value pairs to memorize, followed by a sequence of queries. For each query, the model must retrieve the corresponding key from memory and accurately recall its associated value. Attention based layers perform the best in this task, while



(a) Accuracy vs. model dimension for different fading memory layers on MQAR.

(b) Runtime of a single memory layer

Figure 2. **MQAR results** (a) Each plot corresponds to a particular sequence length and number of key-value pairs for the model to memorize. **Runtime** (b) Runtimes are for a single forward + backward pass (8 heads, head dim 128, batch size 4, averaged over 20 runs).

SSM-based memory layers are known to struggle as their memory fades away as the context length grows.

We compare GKA with Attention and other linear SSM baselines on this task. For each memory layer type, we train 2-layer models on MQAR training data and evaluate on a held-out test set. We repeat this experiment for 4 different learning rates spanning from 10^{-4} to 10^{-2}. As shown in Fig. 2a, GKA improves upon every other linear SSM baseline at all sequence lengths and model dimensions considered. Note, the complexity of the task increases with increasing sequence length and number of key-value pairs, while larger model dimensions improve memorization capacity through increased state size. The success of our layer can be attributed to our modeling choice: unlike other fading memory designs (like GDN or Mamba2), we construct states based on the optimal MAP estimate conditioned on the entire history, enabling better retention of remote information.

5.2. Training Throughput of GKA

In Fig. 2b we measure the running times (forward + backward) of a single GKA layer together with FlashAttention [19], DeltaNet, GDN (see also Section H). Despite having a more computationally expensive state update equation (CH), GKA has comparable running time to GDN. Note that we compared GDN at equal state size (GDN expands value dimension by 2 by default).

5.3. GKA on Language Modeling

5.3.1. Short-context Tasks

Setup. For this set of experiments, we construct 2.8B LLM models for each memory layers (GKA and baselines described in Section 5) by cascading blocks of `mem` + Multi-Layer Perceptron (MLP) blocks.[2] Hereby, we refer to the 2.8B models with the same name as the layer used to construct them. We then train each model on DCLM [35], a generic pre-training dataset for 100B tokens at 4K context length using the AdamW optimizer with a peak Learning Rate (LR) of 10^{-3} and gradient clipping of 1.0. We used the cosine LR scheduler with a warmup period of 5B tokens with a global batch size of 2M tokens. All models employ the GPT2 tokenizer with a vocabulary size of 50K tokens.

Tasks. Following prior works [68, 71, 75], to consider language modeling capabilities of our model we perform zero-shot evaluation on the following eight common-sense reasoning tasks from LM-Harness [16]: Arc-E, Arc-C, BoolQ, COPA, HellaSWAG, PIQA, SciQ, Winogrande. We also evaluate models on FDA and SWDE, real-world recall-intensive tasks which focus on extracting structured information like tagged content from raw text (for example, HTML files). All these tasks are relatively short ($<$ 2K tokens).

Results. We report our results in Table 2. GKA outperforms all fading memory baselines on average across all tasks owing to its ability to better manage its state via solving (3). In particular, GKA outperforms both GDN and Mamba2 on recall-intensive tasks (FDA and SWDE) by about 10% (rel. improvement). We note that although GKA improves upon existing SSM layers there is still a gap with Attention-based Transformer especially on recall-tasks owing to the eidetic capabilities of Attention. Nevertheless, as discussed in Section 1 this improvement comes at a quadratic cost at training time, whereas our layer's computational complexity is still comparable to existing SSM layers (cf. Section 5.2). In Section M we extend our results to Hybrid models (stack of SSM and Attention layers) and show that the gap with full Transformer models becomes negligible (while still benefiting the SSM's computational advantages). Finally, in Section I we show that GKA exhibits stronger scaling with compute than other SSM baseline models.

5.3.2. Long-context Tasks

Setup. To enable long-context capabilities of our models, as is common practice, we perform continued pre-training of our 2.8B models obtained in Section 5.3.1 on 25B tokens of long documents at 128K context length (cf. Appendix). To the best of our knowledge we are the first to train and evaluate SSM models up to 128K context (e.g., previous work [71] only considered up to 4K/8K context).

Tasks. For long-context, we refrain from using perplexity as it is known to have limitations at assessing long-context performance of LLMs [15, 17, 44]. Instead, we turn to recently proposed benchmarks that mix synthetic and real

[2] For Mamba2 baseline, we consider cascading blocks of Mamba2 layer alone since a single Mamba2 layer has the Mamba2 SSM and MLP.

Table 2. **On average GKA improves upon all fading memory baselines across all tasks.** We report results for zero-shot evaluation of 2.8B language models for short-context tasks. For each task, bold indicates highest value followed by underlined.

Model	ARC-C acc_n ↑	ARC-E acc_n ↑	BoolQ acc ↑	COPA acc ↑	HellaSWAG acc_n ↑	PIQA acc_n ↑	SciQ acc_n ↑	Winogrande acc ↑	FDA contains ↑	SWDE contains ↑	Avg
Transformer	32.25	56.10	**64.28**	80.00	60.96	73.56	79.50	61.72	**58.53**	**72.28**	**63.92**
Gated Linear Attention	27.82	50.80	52.57	78.00	48.83	70.13	69.60	54.54	2.81	20.43	47.55
DeltaNet	**32.85**	58.16	42.51	81.00	61.13	73.78	43.90	61.72	11.80	46.08	51.29
Mamba2	32.24	59.64	58.72	82.00	62.23	73.78	79.80	62.19	7.71	41.13	55.94
Gated DeltaNet	32.59	**60.02**	62.75	82.00	62.80	74.32	80.60	62.35	8.26	44.28	57.00
Gated KalmaNet (Ours)	32.51	59.89	61.68	**85.00**	**63.84**	**74.81**	**83.20**	**64.17**	12.89	50.95	58.89

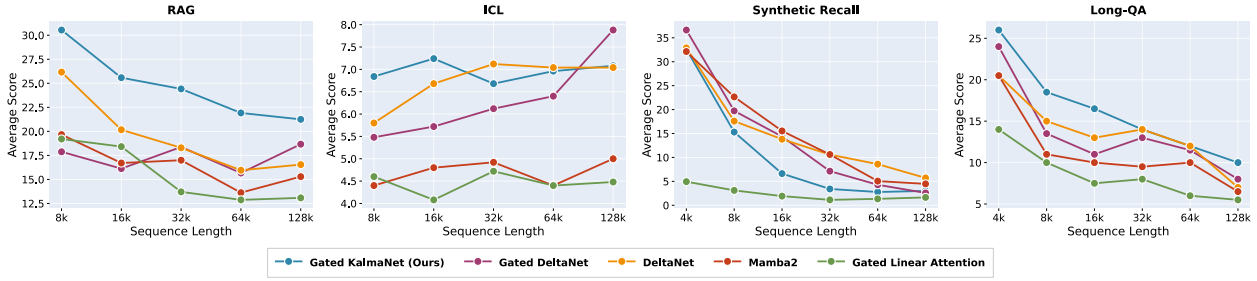


Figure 3. **Long Context Performance up to 128k tokens**. GKA outperforms baselines by 10% in relative improvement on RAG and LongQA. There is no clear winner on Synthetic Recall. All models struggle to perform better than random chance on ICL.

datasets comprising several long-context tasks: Synthetic Recall, Retrieval-Augmented Generation (RAG), Many shot In-Context Learning (ICL) and Long Question-Answering (LongQA). For Synthetic Recall and LongQA we consider tasks from the RULER benchmark [27]. For RAG and ICL we consider tasks from HELMET [72].

Results. Fig. 3 reports our results. GKA shows strong RAG and LongQA capabilities, outperforming all fading memory baselines by at least 10% (rel. improvement). Interestingly, on synthetic recall tasks from RULER, GKA is competitive only at 4K context length and starts to fall behind afterwards. We attribute this divergence to the fundamental differences between these task types. While both RAG and LongQA can be thought of as finding relevant information in long streams of text, they involve more realistic linguistic patterns and semantic relationships that align with natural text distributions seen during pretraining. In contrast, synthetic recall tasks require models to find specific words, numbers, or UUIDs verbatim from long contexts filled with random distractors. This artificial setting does not reflect natural text distributions. Since GKA computes MAP estimates of the latent state based on learned representations of observed tokens, it relies on its pretrained weights to determine which information is important to retain. The synthetic and unnatural structure of Recall tasks makes it difficult for the model to identify what should be retained, as pretrained knowledge provides little signal about importance in these artificial contexts. This suggests that our approach excels in realistic scenarios where pretrained knowledge about natural language structure can guide information selection, but struggles when the signal-to-noise distinction is purely artificial (see Section G for further analysis).

6. GKA on Image Classification

SSMs have gained popularity in vision tasks [24, 37, 77]. Here, we investigate GKA as a viable alternative to other SSMs for vision problems. Specifically, we consider the ImageNet classification task [12] and MambaVision [24], a state-of-the-art Hybrid Mamba-Transformer vision model which modifies the vanilla Mamba layer with (non-causal) convolution operations and a symmetric non-SSM branch, resulting in about 2% improvement [24, Table 4]. We introduce our variant, GKAVision, obtained by replacing the MambaVisionMixer blocks in MambaVision with our GKA layer. More details are provided in Section N. Our results in Table 3, show that GKAVision outperforms MambaVision while closing the gap with a pure vision Transformer (NextViT-S [34]) at 33% higher throughput. We did not implement any vision-specific changes to GKA, which might result in further gains. We conclude the paper in Section B.

Table 3. **GKAVision outperforms MambaVision on ImageNet classification** (averaged over 5 seeds). Model sizes are ≈31.8M. Throughput is measured on 8 H200 GPUs with a batch size of 256.

Model	Top-1 Accuracy (%) ↑	Throughput (K img/s) ↑
MambaVision-T	81.18	**16.25**
GKAVision-T	81.27	13.72
NextViT-S	**81.99**	10.32

References

[1] Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023. 2, 6, 12, 23

[2] Sangmin Bae, Bilge Acun, Haroun Habeeb, Seungyeon Kim, Chien-Yu Lin, Liang Luo, Junjie Wang, and Carole-Jean Wu. Hybrid architectures for language models: Systematic analysis and design insights. *arXiv preprint arXiv:2510.04800*, 2025. 12

[3] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024. 12

[4] Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology*, 12(5): 1–32, 2021. 2

[5] Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025. 12

[6] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery. *arXiv preprint arXiv:2410.05080*, 2024. 12

[7] Hao Cui, Zahra Shamsi, Gowoon Cheon, Xuejian Ma, Shutong Li, Maria Tikhanovskaya, Peter Norgaard, Nayantara Mudur, Martyna Plomecka, Paul Raccuglia, et al. Curie: Evaluating llms on multitask scientific long context understanding and reasoning. *arXiv preprint arXiv:2503.13517*, 2025. 12

[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 2978–2988, 2019. 12

[9] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024. 1, 2, 4, 6, 12

[10] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024. 12

[11] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. 12

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009. 8

[13] Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, et al. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024. 12

[14] Hanpei Fang, Sijie Tao, Nuo Chen, Kai-Xin Chang, and Tetsuya Sakai. Do large language models favor recent content? a study on recency bias in llm-based reranking. *arXiv preprint arXiv:2509.11353*, 2025. 4

[15] Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is wrong with perplexity for long-context language modeling? *arXiv preprint arXiv:2410.23771*, 2024. 7

[16] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024. 2, 7

[17] Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7376–7399, 2025. 7

[18] Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024. 12

[19] Alicia Golden, Samuel Hsia, Fei Sun, Bilge Acun, Basil Hosmer, Yejin Lee, Zachary DeVito, Jeff Johnson, Gu-Yeon Wei, David Brooks, et al. Is flash attention stable? *arXiv preprint arXiv:2405.02803*, 2024. 7

[20] Gene H Golub and Charles F Van Loan. *Matrix Computations (4th ed.)*. The Johns Hopkins University Press, 2013. 3

[21] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. 12

[22] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396, 2021. 12

[23] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021. 12

[24] Ali Hatamizadeh and Jan Kautz. Mambavision: A hybrid mamba-transformer vision backbone. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 25261–25270, 2025. 8, 30

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 2

[26] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de

Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. An empirical analysis of compute-optimal large language model training. *Advances in neural information processing systems*, 35:30016–30030, 2022. 25

[27] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024. 2, 8, 23

[28] Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*, 2024. 12

[29] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. 12

[30] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. 1, 12

[31] Gregory Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. 23

[32] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020. 12

[33] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 12

[34] Jiashi Li, Xin Xia, Wei Li, Huixia Li, Xing Wang, Xuefeng Xiao, Rui Wang, Min Zheng, and Xin Pan. Next-vit: Next generation vision transformer for efficient deployment in realistic industrial scenarios. *arXiv preprint arXiv:2207.05501*, 2022. 8

[35] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024. 7

[36] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. 12

[37] Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, et al. Longhorn: State space models are amortized online learners. In *The Thirteenth International Conference on Learning Representations*. 8

[38] Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In *International Conference on Learning Representations*, 2025. 1, 2

[39] Stefan Ljung and Lennart Ljung. Error propagation properties of recursive least-squares adaptation algorithms. *Automatica*, 21(2):157–167, 1985. 12

[40] Mark D McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton van den Hengel. RanPAC: Random projections and pre-trained models for continual learning. *Advances in Neural Information Processing Systems*, 2023. 3

[41] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*, 2023. 12

[42] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101, 2024. 12

[43] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964. 2

[44] Elvis Nunez, Luca Zancato, Benjamin Bowman, Aditya Golatkar, Wei Xia, and Stefano Soatto. Expansion span: Combining fading memory and retrieval in hybrid state space models. *arXiv preprint arXiv:2412.13328*, 2024. 7, 12

[45] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR, 2023. 12

[46] Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms. *arXiv preprint arXiv:2411.19379*, 2024. 12

[47] Fabian Pedregosa. Residual polynomials and the Chebyshev method. http://fa.bianp.net/blog/2020/polyopt/, 2020. 4

[48] Bo Peng, Daniel Goldstein, Quentin Gregory Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Kranthi Kiran GV, Haowen Hou, Satyapriya Krishna, Ronald McClelland Jr., Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Jian Zhu, and Rui-Jie Zhu. Eagle and finch: RWKV with matrix-valued states and dynamic recurrence. In *Conference on Language Modeling*, 2024. 2

[49] Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. Technical report, arXiv preprint arXiv:2503.14456, 2025. 2

[50] Liangzu Peng and René Vidal. Mathematics of continual learning. Technical report, arXiv:2504.17963 [cs.LG], 2025. 3

[51] Liangzu Peng, Aditya Chattopadhyay, Luca Zancato, Elvis Nunez, Wei Xia, and Stefano Soatto. Gated KalmaNet: A fading memory layer through test-time ridge regression. Technical report, arXiv:2511.21016 [cs.LG], 2025. 3

[52] Liangzu Peng, Juan Elenter, Joshua Agterberg, Alejandro Ribeiro, and Rene Vidal. TSVD: Bridging theory and practice in continual learning with pre-trained models. In *International Conference on Learning Representations*, 2025. 3

[53] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. Kilt: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, 2021. 23

[54] A.H. Sayed. *Fundamentals of Adaptive Filtering*. Wiley, 2003. 12

[55] A.H. Sayed. *Adaptive Filters*. Wiley, 2011. 12

[56] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, 2021. 2

[57] Samir Shah, Francesco Palmieri, and Michael Datum. Optimal filtering algorithms for fast learning in feedforward neural networks. *Neural Networks*, 5(5):779–787, 1992. 3

[58] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. Technical report, arXiv:2502.10297v6 [cs.LG], 2025. 2

[59] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. Technical report, arXiv:2307.08621v4 [cs.CL], 2023. 2

[60] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023. 12

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 1, 2, 12

[62] René Vidal. Attention: Self-expression is all you need, 2022. 2

[63] Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. Mesanet: Sequence modeling by locally optimal test-time training. Technical report, arXiv:2506.05233 [cs.LG], 2025. 3, 21, 26

[64] Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024. 12

[65] Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. Technical report, arXiv:2501.12352v3 [cs.LG], 2025. 3

[66] Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964. 2

[67] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 6, 29

[68] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, 2024. 1, 2, 4, 6, 7, 12

[69] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, pages 115491–115522. Curran Associates, Inc., 2024. 12

[70] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Neural Information Processing Systems*, 2024. 1, 2, 3, 4, 6

[71] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. 2, 6, 7, 12

[72] Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*, 2025. 2, 8, 23

[73] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, 2025. 12

[74] Luca Zancato, Alessandro Achille, Giovanni Paolini, Alessandro Chiuso, and Stefano Soatto. Stacked residuals of dynamic layers for time series anomaly detection. *arXiv preprint arXiv:2202.12457*, 2022. 12

[75] Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'mojo: Hybrid state space realizations of foundation models with eidetic and fading memory. In *Advances in Neural Information Processing Systems*, pages 130433–130462. Curran Associates, Inc., 2024. 1, 7, 12

[76] Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019. 3

[77] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. In *Forty-first International Conference on Machine Learning*. 8