

Kimi Linear & Kimi Delta Attention

arXiv: 2510.26692v2

David MacMillan
Deep Learning Study Group
11/11/2025 Meetup

ABSTRACT

We introduce Kimi Linear, a hybrid linear attention architecture that, for the first time, outperforms full attention under fair comparisons across various scenarios—including short-context, long-context, and reinforcement learning (RL) scaling regimes. At its core lies Kimi Delta Attention (KDA), an expressive linear attention module that extends Gated DeltaNet [111] with a finer-grained gating mechanism, enabling more effective use of limited finite-state RNN memory. Our bespoke chunkwise algorithm achieves high hardware efficiency through a specialized variant of the *Diagonal-Plus-Low-Rank* (DPLR) transition matrices, which substantially reduces computation compared to the general DPLR formulation while remaining more consistent with the classical delta rule.

We pretrain a Kimi Linear model with 3B activated parameters and 48B total parameters, based on a layerwise hybrid of KDA and Multi-Head Latent Attention (MLA). Our experiments show that with an identical training recipe, Kimi Linear outperforms full MLA with a sizeable margin across all evaluated tasks, while reducing KV cache usage by up to 75% and achieving up to 6× decoding throughput for a 1M context. These results demonstrate that Kimi Linear can be a drop-in replacement for full attention architectures with superior performance and efficiency, including tasks with longer input and output lengths.

To support further research, we open-source the KDA kernel and vLLM implementations ¹, and release the pre-trained and instruction-tuned model checkpoints. ²

Ancestry of Kimi Linear's changes to Attention

- **Kimi Linear (with Kimi Delta Attention) - combines many concepts, including: ([] is ref # in paper)**
 - **Concept 1: Linearizing Attention**
 - [48] "Transformers are RNNs" - DLSG 2024-06-05
<https://dl.acm.org/doi/pdf/10.5555/3524938.3525416>
YouTubes: Yannic has one and also two from MIT
 - **Concept 2: Delta Rule and Fast Weights**
 - "Adaptive Switching Circuits" - Adaline paper - Widrow & Hoff 1960
 - [84] "Linear Transformers are Secretly Fast Weight Programmers" - arXiv: 2102.11174 Schlag, Irie, Schmidhuber
 - [112] "Parallelizing Linear Transformers with the Delta Rule" - arXiv 2406.06484 Songlin Yang et al
 - **Concept 3: Gating or Decay**
 - [111] "Gated Delta Networks: Improving Mamba 2" - arXiv:2412.06464 Songlin Yang et al
 - [71] ' RWKV-7 "Goose" with Expressive Dynamics State Evolution' - arXiv: 2503.14456
 - [92] "Retentive Network" - arXiv:2307.08621
 - [16] "Transformers are SSMs" - DLSG 2024-06-18 - arXiv: 2405.21060
 - [52] "Transformers are RNNs" --> see Concept 1, above
 - [114] "Gated Linear Attention Transformers" - arXiv: 2312.06635

Background for Concept 1: Linearizing Attention

The Kernel Trick

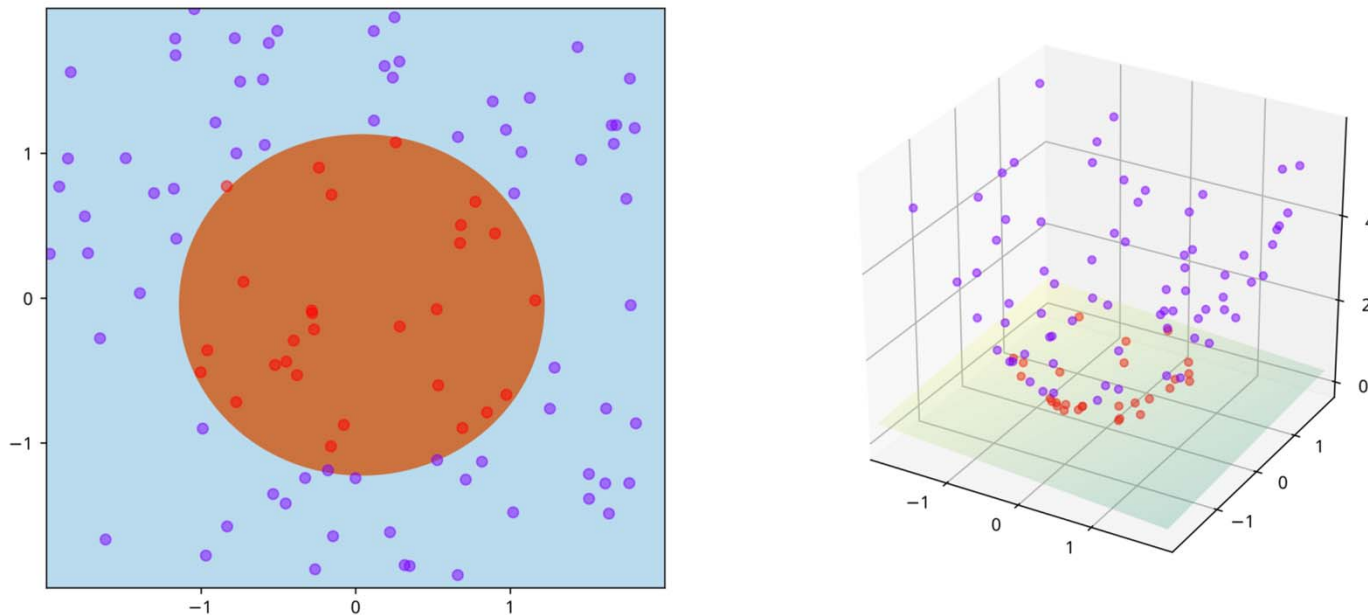


Image by Shiyu Ji - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=60458994>
from https://en.wikipedia.org/wiki/Kernel_method

Also see:

- <https://www.youtube.com/watch?v=hAooAOFRsYc> at 16:00
- https://www.youtube.com/shorts/O__dpIFy390?app=desktop

Concept 1: Linearizing Attention - from [48] "Transformers are RNNs"

- Standard attention

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \\ A_l(x) &= V' = \text{softmax} \left(\frac{QK^T}{\sqrt{D}} \right) V. \end{aligned} \tag{2}$$

- Can generalize this - instead of softmax, use an arbitrary similarity function: sim()

$$V'_i = \frac{\sum_{j=1}^N \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^N \text{sim}(Q_i, K_j)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with $\text{sim}(q, k) = \exp \left(\frac{q^T k}{\sqrt{D}} \right)$.

Concept 1: Linearizing Attention - from [48] "Transformers are RNNs"

- Use kernel trick with $\phi()$ to linearize. Note it is applied rowwise to matrices Q and K.

Simplify:

(4)→(5) Sum is not over i, so
move $\phi(Q_i)^T$ outside

In (5) Vectors V_j^T aggregated by
red box's similarity terms
(i.e. red box routes the V vectors)

In (5) denominator normalizes

In (6) refactors (5) to RHS
Now just calc blue box once,
then use that for each Q

So we are now we are
 $O(N)$ vs. softmax $O(N^2)$

Given such a kernel with a feature representation $\phi(x)$ we can rewrite equation 2 as follows,

$$V'_i = \frac{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^N \phi(Q_i)^T \phi(K_j)}, \quad (4)$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V'_i = \frac{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^N \phi(K_j)}. \quad (5)$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$\left(\phi(Q) \phi(K)^T \right) V = \phi(Q) \left(\phi(K)^T V \right). \quad (6)$$

Note that the feature map $\phi(\cdot)$ is applied rowwise to the matrices Q and K.

See Yannic's video at 20:00 for more details

Concept 2: Delta Rule - from Widrow & Hoff, 1960

- Origin seems to be "Adaptive Switching Circuits", Bernard Widrow & Marcian Hoff, 1960, or possibly R. L. Mattson's Master's Thesis at MIT (1959), who Widrow-Hoff cite as [5],[6].
<https://www-isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf>

Historical notes:

- This "Adaptive Switching Circuits" paper is the famous "Adaline" (adaptive linear) system
- Bernard Widrow was a major neural net researcher, building on Rosenblatt and Von Neumann. At MIT he worked on core memory for Whirlwind 1 (a tube-based computer).
- Marcian "Ted" Hoff later co-invented the microprocessor as employee #12 at Intel. He wanted a "universal processor" (4004 CPU 1971) to avoid making a variety of custom-designed circuits.
- Closing paragraph of the Woodrow & Hoff paper:
 - "Very sophisticated learning procedures would become possible if one had such recall-by-association parallel-access memory systems. The simplicity of Adaline and the progress being made in microelectronics gives a strong indication that such memory systems will come into existence in the not too distant future" (1960)
- Intel's first chip was the 1103, a digital (not analog) 1kbit DRAM (1970)

Adaline, an NVidia rack and Stargate (Abilene) (just one of many)

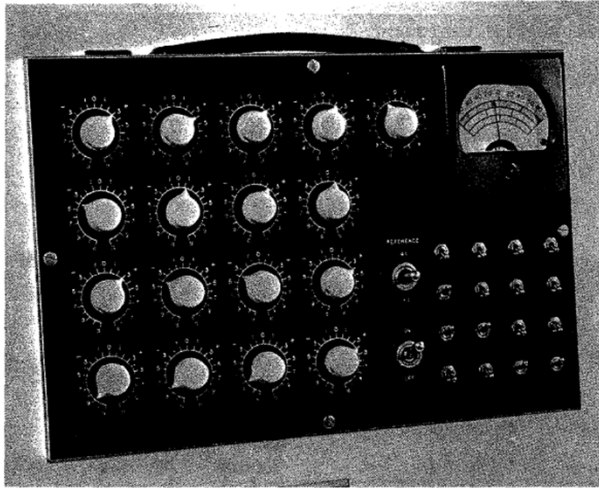
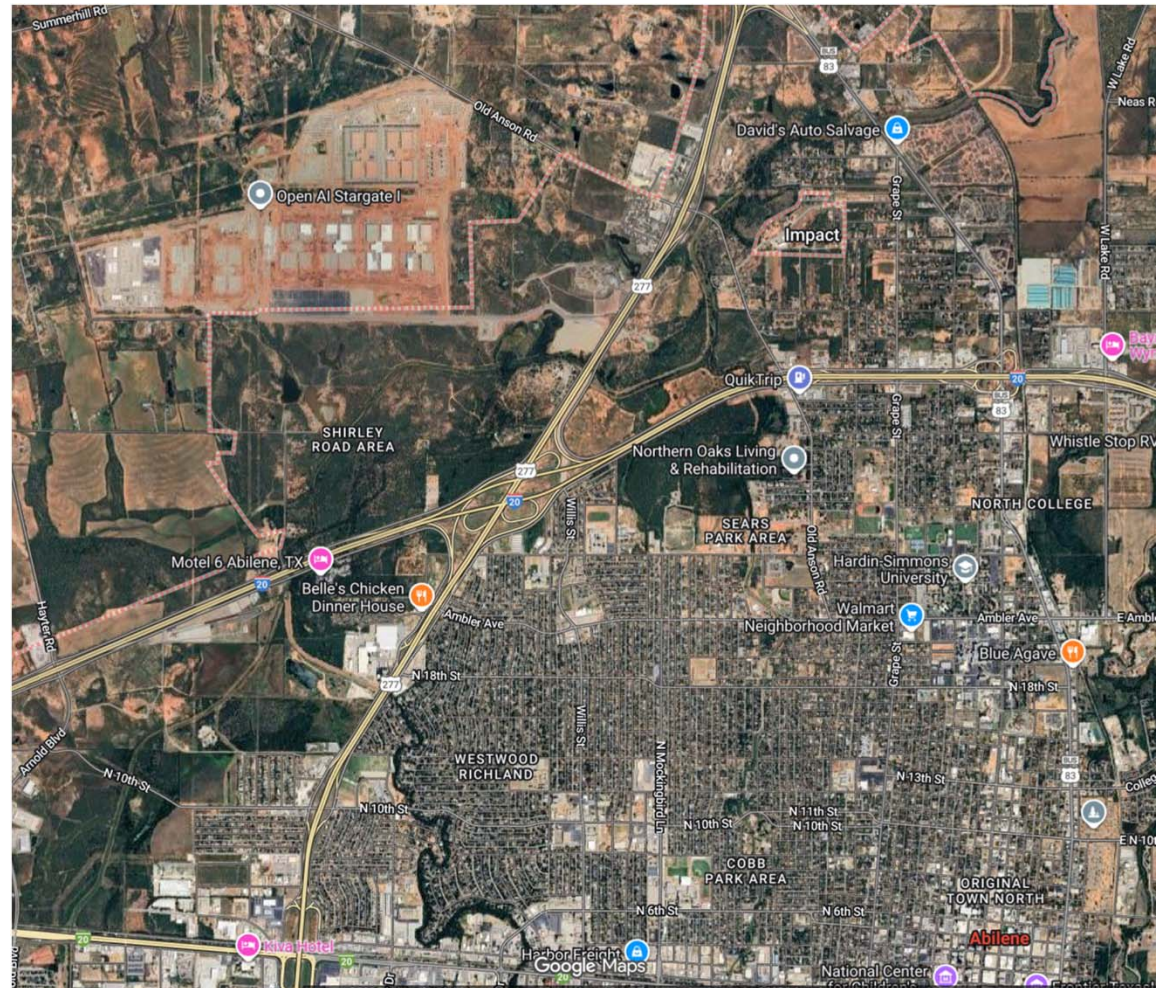
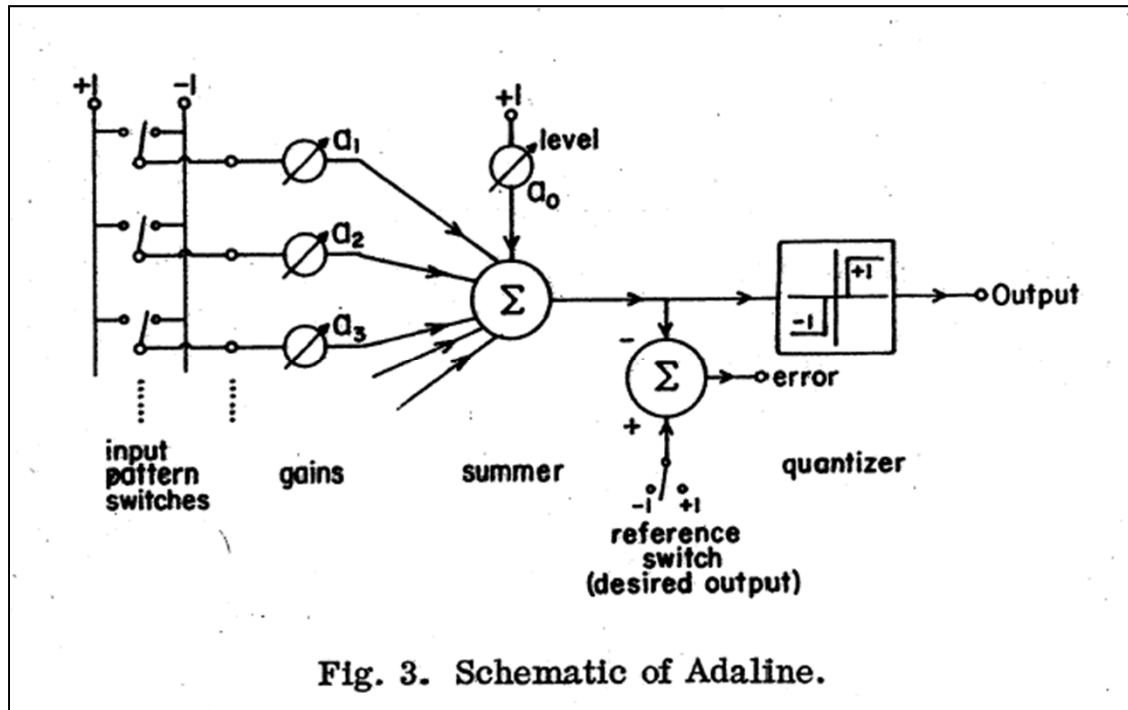


Fig. 2. Adaline.



Adaptive Switching Circuits - Adaline



- Can learn several categories (e.g. A,B,C) by dividing range on meter (-60=A, 0=B, -60=C).
- Supervised training procedure (delta rule).
 - Set toggle switches on inputs and for desired output (17 in all).
 - Adjust all weights by equal absolute magnitude to bring error to zero.
 - Repeat with next training data.
 - Converges rapidly to small fluctuations around a root-mean-squared value.
- Paper discusses the constraint of needing linear separable patterns (our Concept 1)
- Delta rule is sequential (remember this for what comes later)

Concept 2 - The Delta Rule - Fast weights

- "Normal" or "slow" weights are the matrices adjusted by gradient descent & backprop
- Activations can be viewed as "fast weights"
- Schlag et al "Linear Transformers are Secretly Fast Weight Programmers" 2102.11174
 - "We show the formal equivalence of linearised self-attention mechanisms and fast weight controllers from the early '90s, where a ``slow" neural net learns by gradient descent to program the ``fast weights" of another net through sequences of elementary programming instructions which are additive outer products of self-invented activation patterns (today called keys and values)."
 - "Such Fast Weight Programmers (FWPs) learn to manipulate the contents of a finite memory and dynamically interact with it."
 - "We infer a memory capacity limitation of recent linearised softmax attention variants, and replace the purely additive outer products by a delta rule-like programming instruction, such that the FWP can more easily learn to correct the current mapping from keys to values. The FWP also learns to compute dynamically changing learning rates. "
 - Can replace value at current key with same amount of new value, for per-key memory replacement [c.f. 2503.14456]
 - "We also propose a new kernel function to linearise attention which balances simplicity and effectiveness."

Concepts 2 & 3 - The Delta Rule with Decay

- Songlin Yang et al "Gated Delta Networks: Improving Mamba2 with Delta Rule" 2412.06464
- "... the [transformer] self-attention component scales quadratically with sequence length, leading to substantial computational demands ..."
- "To mitigate these issues, researchers have explored alternatives such as linear Transformers (Katharopoulos et al., 2020a), which replace traditional softmax-based attention with kernelized dot-product-based linear attention, substantially reducing memory requirements during inference by reframing as a linear RNN with matrix-valued states. ... However, challenges persist in managing information over long sequences, particularly for in-context retrieval tasks..."
- " the number of orthogonal key-value pairs they [linear transformers] can store is bounded by the model's dimensionality. When the sequence length exceeds this dimension, "memory collisions" become inevitable, hindering exact retrieval (Schlag et al., 2021a).
- Mamba2 addresses this limitation by introducing a simple gated update rule ... which uniformly decays all key-value associations at each time step by a dynamic ratio $\alpha \in (0, 1)$

Need for Gating

- Songlin Yang et al "Gated Delta Networks: Improving Mamba2 with Delta Rule" 2412.06464
- "However, this approach does not account for the varying importance of different key-value associations, potentially leading to inefficient memory utilization. If the model needs to forget a specific key-value association, all key-value associations are equally forgotten, making the process less targeted and efficient."
- "In contrast, the linear Transformer with the delta rule (Widrow et al., 1960), known as DeltaNet (Schlag et al., 2021a; Yang et al., 2024b), selectively updates memory by (softly) replacing an old key-value pair with the incoming one in a sequential manner. This method has demonstrated impressive performance in synthetic benchmarks for in-context retrieval."
- "However, since this process only modifies a single key-value pair at a time, the model lacks the ability to rapidly clear outdated or irrelevant information, especially during context switches where previous data needs to be erased."
- Consequently, DeltaNet has been found to perform moderately on real-world tasks (Yang et al., 2024b), likely due to the absence of a robust memory-clearing mechanism."

Gated Delta

- Songlin Yang et al "Gated Delta Networks: Improving Mamba2 with Delta Rule" 2412.06464
- "Recognizing the complementary advantages of the gated update rule and the delta rule in memory management, we propose the gated delta rule, a simple and intuitive mechanism that combines both approaches. This unified rule enables flexible memory control: it can promptly clear memory by setting $\alpha \rightarrow 0$, while selectively updating specific content without affecting other information by setting $\alpha t \rightarrow 1$ (effectively switching to the pure delta rule)."
- "... challenge lies in implementing the gated delta rule in a hardware-efficient manner."
- "Building upon Yang et al. ([NeurIPS] 2024b)'s efficient algorithm that parallelizes the delta rule computation using the WY representation (Bischof & Loan, 1985), we carefully extend their approach to incorporate the gating terms. Our extension preserves the benefits of chunkwise parallelism (Hua et al., 2022b; Sun et al., 2023a; Yang et al., 2024a;b), enabling hardware-efficient training."
 - Gating is by a scalar per head.
- The WY representation optimizes performance by doing one matrix multiplication instead of a sequential series of matrix multiplications.
 - It expresses a product of Householder matrices $Q=P_1 P_2 \dots P_r$ in the form $Q=I + WY^T$
- Householder matrices are orthogonal (so square), symmetric and involutory (is its own inverse)

The Need for Chunking

- **First some terminology:**
 - **Training** - what usually is referred to as 'supervised pre-training' where you learn your weights
- **Inference** - has two stages
 - **Pre-fill** - when you drop the prompt in and want to calculate the next token from the full initial prompt
 - **Decoding** - what happens after you get the first token output from the pre-fill. The process of repeatedly generating a new output token, adding it back to the input context, and then repeating this decoding process over & over (aka autoregression).
- **The delta update rule requires tracking of the state from one token to the next.**
 - Fits naturally during decoding (autoregression)
 - But during training and pre-fill, naive implementation would require sequentially processing one token at a time, working through to the state after the last token.
 - This is way too slow and would make the delta rule impractical.
 - Chunking is a way to drastically speed this up by doing a bunch of tokens all at once.

Processing L/C Chunks: With vs Without Chunking

Aspect	Without Chunking (Naive)	With Chunking (WY Representation)
Total tokens	L tokens	L tokens
Sequential steps	L steps (one per token)	L/C steps (one per chunk)
Parallelism	None - fully sequential	C tokens processed in parallel per chunk
Example (L=4096, C=64)	4,096 sequential steps	64 sequential chunks
Per-step operation	$S_t = (I - \beta_t \cdot k_t \cdot k_t^T) \cdot S_{t-1} + \beta_t \cdot k_t \cdot v_t^T$	Apply WY-compacted transform for 64 tokens at once
Computation type	Many small rank-1 matrix updates	Few large dense matrix multiplications
GPU utilization	Poor (sequential, small ops)	Excellent (parallel, dense matmul)
Memory per step	State S (fixed size)	State S + temporary W, U matrices
Speedup	Baseline (1x)	~Cx faster (~64x in practice)
State updates	$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_L$	$S_0 \rightarrow S_{64} \rightarrow S_{128} \rightarrow \dots \rightarrow S_L$

Key insight: Chunking reduces sequential dependencies from **L** to **L/C** by batching C rank-1 updates into parallel-friendly matrix operations using WY representation.

DM Compare a normal transformer's computational requirements to a chunked approach where $L = C$.

Analyzed computational complexity tradeoffs between transformer and chunked approach... 10s ▾

Computational Requirements: Transformer vs. One Giant Chunk ($L = C$)

Formulas

Standard Transformer: $\text{FLOPs_Attn}(T; d_h) = 2T^2 \cdot d_h$ [arXiv](#)

KDA with $L = C$ (one giant chunk): $\text{FLOPs_KDA}(T; C, d_h) = 6T \cdot d_h^2 + 3T \cdot C \cdot d_h + T \cdot C^2$
[arXiv](#)

Substituting $C = T$: **FLOPs** = $6T \cdot d_h^2 + 3T^2 \cdot d_h + T^3$

Concrete Example ($T = 100K$ tokens, $d_h = 128$)

Method	Formula	FLOPs (billions)	Complexity
Transformer	$2T^2 \cdot d_h$	2,560	$O(T^2)$
One Giant Chunk ($C=T$)	$6T \cdot d_h^2 + 3T^2 \cdot d_h + T^3$	1,003,850	$O(T^3)$ 🤯
Chunked ($C=64$)	$6T \cdot d_h^2 + 3T \cdot C \cdot d_h + T \cdot C^2$	12.6	$O(T)$ ✅

The Problem

With one giant chunk, the T^3 term (from $T \cdot C^2$) completely dominates, making it:

- **~392× WORSE than transformer!**
- **~79,670× WORSE than proper chunking!**

Why This Happens

The C^2 term in the WY algorithm represents the cost of computing relationships **within** the chunk. When C = entire sequence, you're computing T^2 relationships, but doing it in a way that costs T^3 - worse than just using a transformer!

Moral: Chunking with small C (like 64) keeps the C^2 term manageable while still reducing sequential steps dramatically. It's the only way to beat transformers.

Concepts 2 & 3 - Fine-Grained Delta Rule in "Goose" paper

- "Generalizes the [Widrow] Delta Rule for use in sequence modeling"
- "Expand in-context learning rate [and gating] from a scalar to become vector-valued, allowing model to selectively replace state data on a channel-wise basis".
 - Implemented with parallelized weighted key value (wkv) transition matrix
 - Can recognize all regular languages (i.e. language can be described by regex).
- Transition matrix not Householder but is scaled approximation that mimics Householder

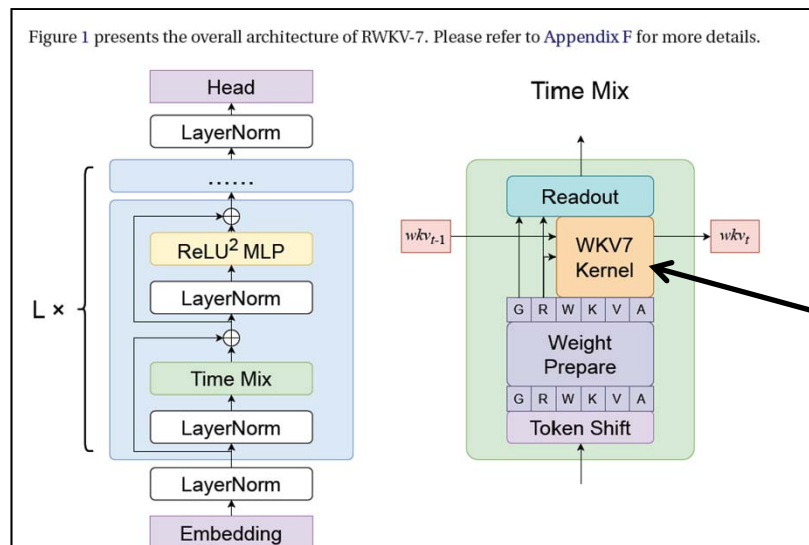


Figure 1: RWKV-7's overall architecture.

$$wkv_t = wkv_{t-1} \left(\text{diag}(w_t) - \kappa_t^T (a_t \odot \kappa_t) \right) + v_t^T \bar{k}_t$$

Figure 2: A simple illustration of the update mechanism of a single head of RWKV-7's state. Note that the actual state size is 64×64 per head, not 4×4 .

c.f. Kimi paper's (1)

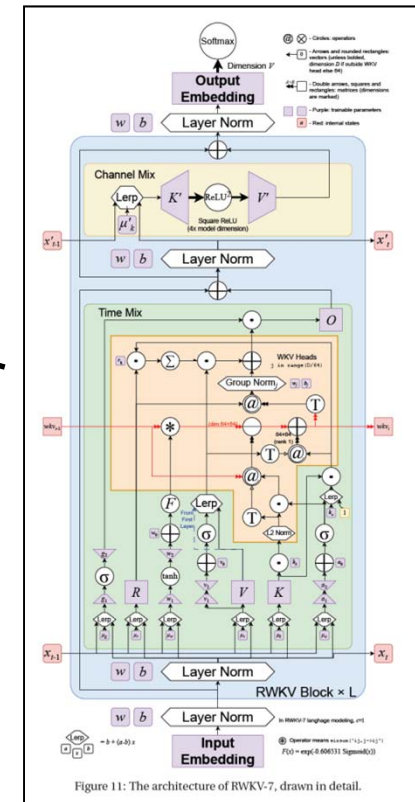


Figure 11: The architecture of RWKV-7, drawn in detail.

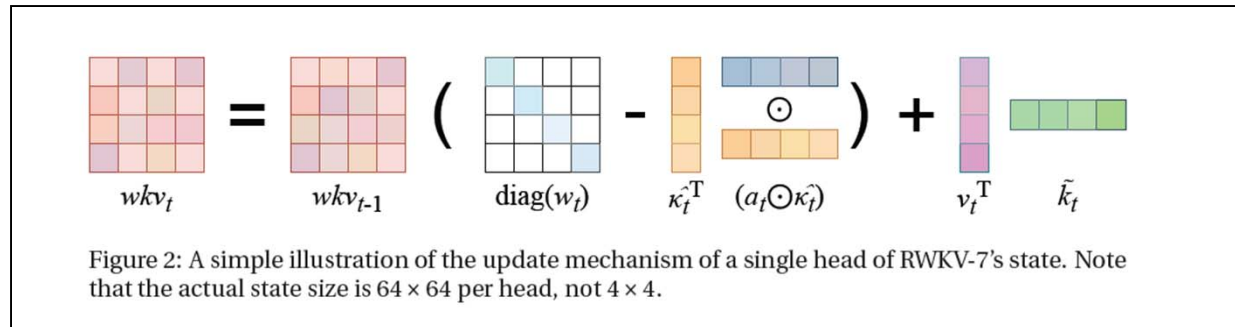
(Just showing there is a lot inside)

Comment

- This is a surprise - they let the model "selectively replace state data on a channel-wise basis"
 - Channels and dimensions are the same in a transformer.
 - An embedding vector is made up of the channels.
 - Recall the male-king, female-queen vector math in Word2Vec
- Wait a minute! They are replacing and decaying on a *per-channel* basis, so they can alter each element of the embedding vector activation at a different rate?
 - This moves the vector in vector space!
 - So doesn't that mess up the male-king, female-queen math??? Isn't this a fatal flaw?
 - Apparently not.
 - Perhaps the network learns decay rates such that all relevant columns to a relationship (like king-queen) decay in lockstep and the other noisy channels are allowed to decay faster or slower.
 - In other words, the training moves the relevant relationships in lockstep and forgets about the other channels.
 - This is speculation on my part - I didn't see anything in the papers addressing this.
- Notably Goose boosted the channel dimension (the embedding) by a factor of 4.
 - So they have more 'room' to play around with embedding sub-spaces.

Fine grained decay

- **Goose paper**



- **Tonight's paper - Kimi Linear**

$$\mathbf{S}_t = (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) \text{Diag}(\alpha_t) \mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top \in \mathbb{R}^{d_k \times d_v}; \quad \mathbf{o}_t = \mathbf{S}_t^\top \mathbf{q}_t \in \mathbb{R}^{d_v} \quad (1)$$

Recap

- Can linearize using kernel trick
- Widrow-Hoff decay (Adaline). It is sequential.
 - Parallelized by Yang et. al Neurips 2024
- Katharopoulos et al., 2020 - Transformer is quadratic so $O(N^2)$
 - Kernelized linear transformers are much faster $O(N)$, but weak in-context retrieval
- Schlag et. al - linearized transformers activations are like fast weights that implement stochastic gradient descent
- Yang 2412.06464 - linear transformers have limited representation due to orthogonality
 - Mamba2 adds uniform decay to address this, but uniformly forgets all key-value pairs
 - But Widrow-Hoff softly adjusts each key-value
 - Yang et. al - Gated delta combines both ideas. Gating is by a scalar per head.
 - Uses WY representation & chunking to avoid Widrow-Hoff sequential calcs
 - Added ability to clear for fast context switches in production environment
- Goose - Per channel gating and learning rate using parallelized wkv transition matrix

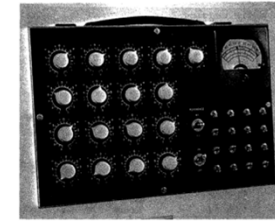
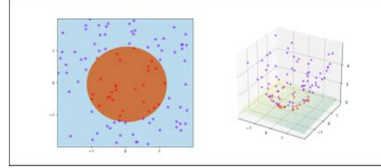


Fig. 2. Adaline.