

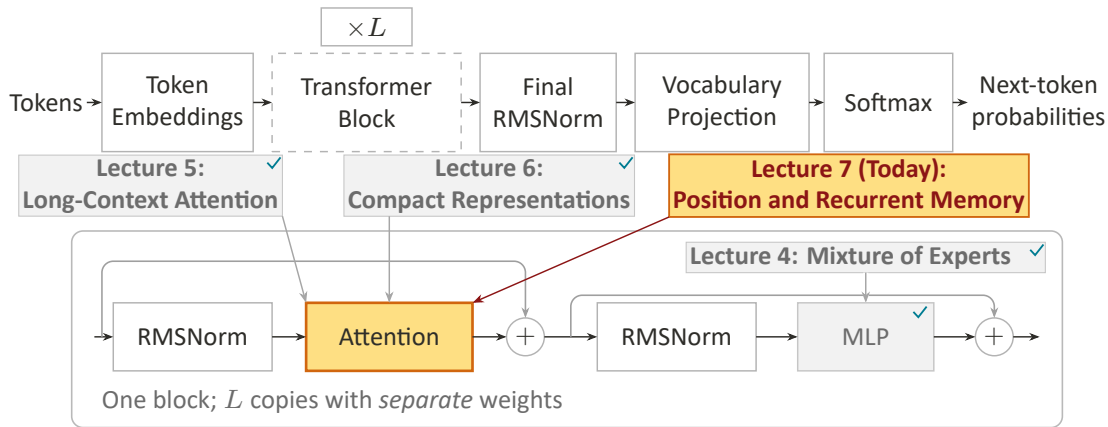
Making Long-Context Attention Efficient

Lecture 7: Positional Embeddings
and Recurrent Memory

Anay Mehrotra
MS&E 319
Autumn 2026



Topic 2: Efficient attention

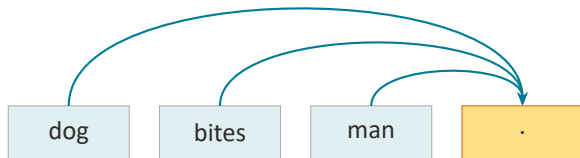


Can we reduce the cost per token while keeping quality fixed?

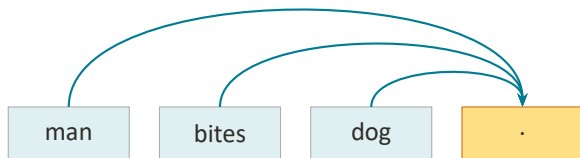


Why attention needs position

Without position, attention cannot distinguish these word orders.

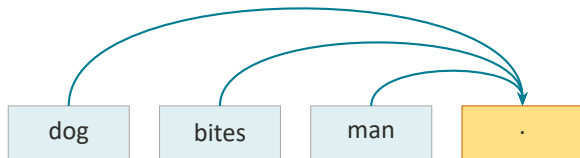


same keys, values, and scores:
the same output for "." in layer 1

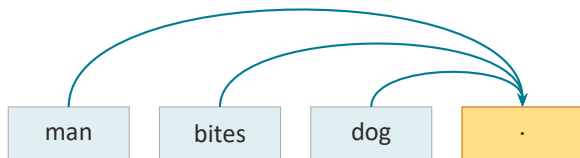


Why attention needs position

Without position, attention cannot distinguish these word orders.



same keys, values, and scores:
the same output for "." in layer 1

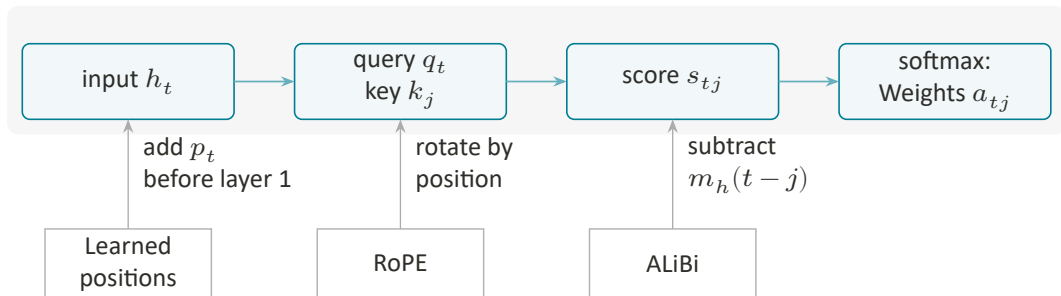


Permuting the key-value pairs leaves this read unchanged.

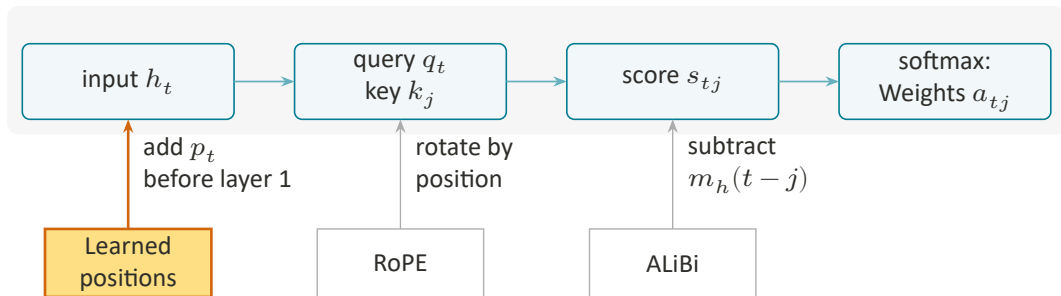
The causal mask gives other tokens different prefixes; deeper layers can differ.



Positional encodings



Positional encodings



What if each position had its own learned vector?

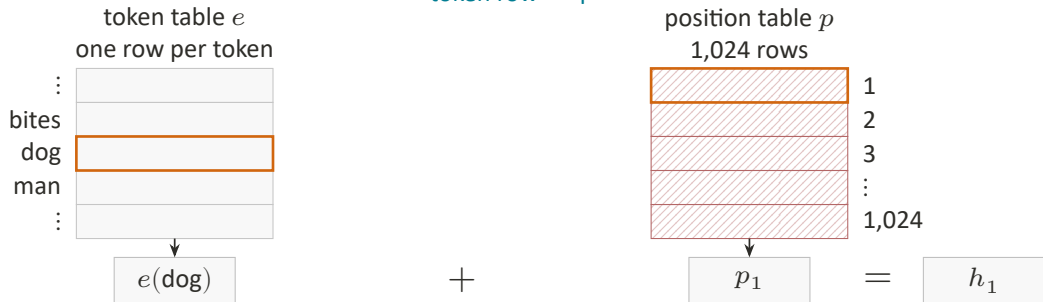


Learned positions

Gehring et al. (2017)

GPT-2 learns two tables and adds one row from each.

$$h_t = \underbrace{e(x_t)}_{\text{token row}} + \underbrace{p_t}_{\text{position row}}$$



Source: Radford et al. (2018, 2019), GPT-1 and GPT-2.

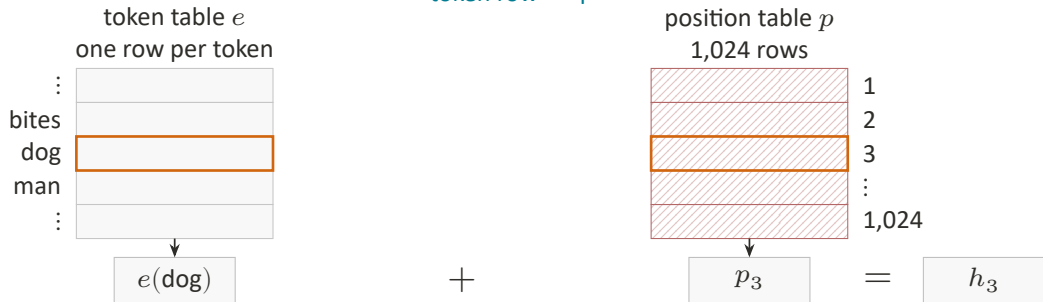


Learned positions

Gehring et al. (2017)

GPT-2 learns two tables and adds one row from each.

$$h_t = \underbrace{e(x_t)}_{\text{token row}} + \underbrace{p_t}_{\text{position row}}$$



Source: Radford et al. (2018, 2019), GPT-1 and GPT-2.

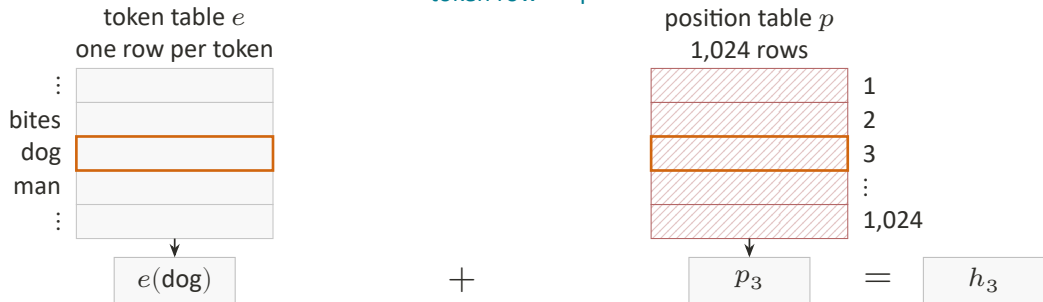


Learned positions

Gehring et al. (2017)

GPT-2 learns two tables and adds one row from each.

$$h_t = \underbrace{e(x_t)}_{\text{token row}} + \underbrace{p_t}_{\text{position row}}$$



The same word gets a different embedding at each position.

Source: Radford et al. (2018, 2019), GPT-1 and GPT-2.



Learned positions inside the score

Layer 1, ignoring scaling and normalization, with $M = W_Q W_K^\top$:

$$q_t^\top k_j = (e_t + p_t)^\top M (e_j + p_j)$$

for each position i :
 $e_i = e(x_i)$



Learned positions inside the score

Layer 1, ignoring scaling and normalization, with $M = W_Q W_K^\top$:

$$q_t^\top k_j = (e_t + p_t)^\top M (e_j + p_j)$$

for each position i :

$$e_i = e(x_i)$$

$$= \underbrace{e_t^\top M e_j}_{\text{word, word}} + \underbrace{e_t^\top M p_j + p_t^\top M e_j}_{\text{word, position}} + \underbrace{p_t^\top M p_j}_{\text{position, position}}$$

The last three terms depend on position.



Learned positions inside the score

Layer 1, ignoring scaling and normalization, with $M = W_Q W_K^\top$:

$$q_t^\top k_j = (e_t + p_t)^\top M (e_j + p_j)$$

for each position i :

$$e_i = e(x_i)$$

$$= \underbrace{e_t^\top M e_j}_{\text{word, word}} + \underbrace{e_t^\top M p_j + p_t^\top M e_j}_{\text{word, position}} + \underbrace{p_t^\top M p_j}_{\text{position, position}}$$

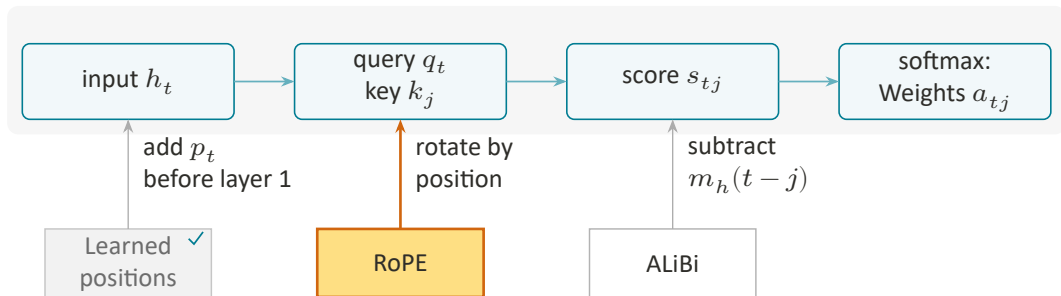
The last three terms depend on position.

Shift both tokens by 5: $p_t^\top M p_j$ becomes $p_{t+5}^\top M p_{j+5}$.

This term can change even though the offset $t - j$ stays the same.



Positional encodings



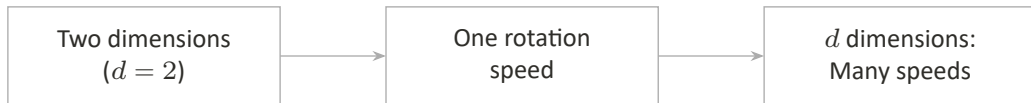
Can the score depend only on the distance between tokens?



Rotary position embedding (RoPE)

Su et al. (2021)

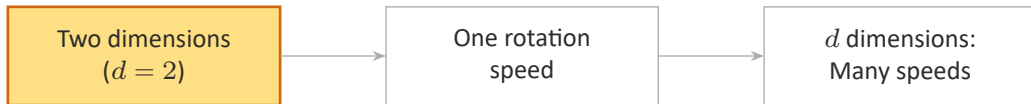
Rotate queries and keys by an angle proportional to position.



Rotary position embedding (RoPE)

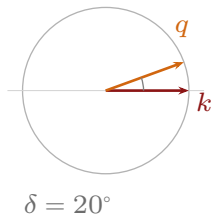
Su et al. (2021)

Rotate queries and keys by an angle proportional to position.



RoPE: How the angle sets the score

Take two dimensions: $q, k \in \mathbb{R}^2$.

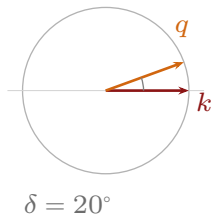


$$q^\top k = \|q\| \|k\| \cos \delta$$



RoPE: How the angle sets the score

Take two dimensions: $q, k \in \mathbb{R}^2$.



$$q^T k = \|q\| \|k\| \cos \delta$$

Simplifying assumption: $\|q\| = \|k\| = 1$

Then the score $q^T k = \cos \delta$ depends only on the angle δ .

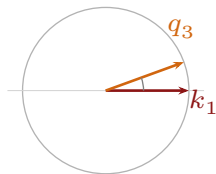


RoPE: Rotate by position

Rotate each vector by an angle proportional to its position.

$$\hat{q}_t = R_t q_t, \quad \hat{k}_j = R_j k_j, \quad R_t : \text{Rotate by } t\theta$$

before rotation



$$\delta = 20^\circ$$



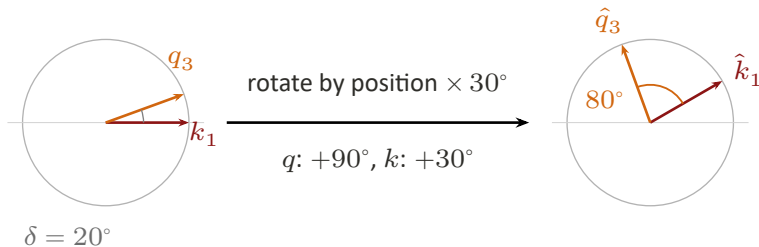
RoPE: Rotate by position

Rotate each vector by an angle proportional to its position.

$$\hat{q}_t = R_t q_t, \quad \hat{k}_j = R_j k_j, \quad R_t : \text{Rotate by } t\theta$$

before rotation

man at 3, dog at 1



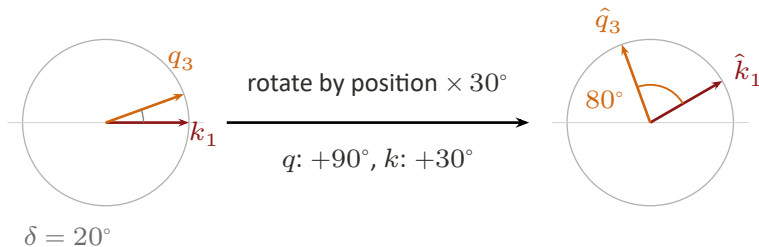
RoPE: Rotate by position

Rotate each vector by an angle proportional to its position.

$$\hat{q}_t = R_t q_t, \quad \hat{k}_j = R_j k_j, \quad R_t : \text{Rotate by } t\theta$$

before rotation

man at 3, dog at 1



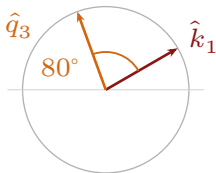
The gap grows by $(t - j)\theta = 60^\circ$.



RoPE: Only the offset matters

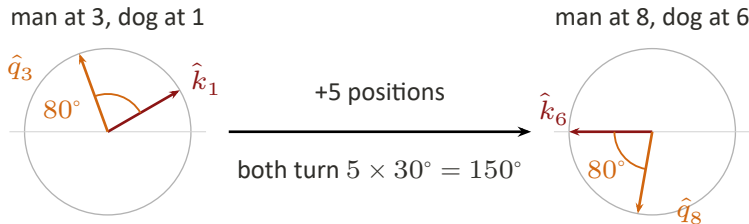
Now move both words five positions later.

man at 3, dog at 1



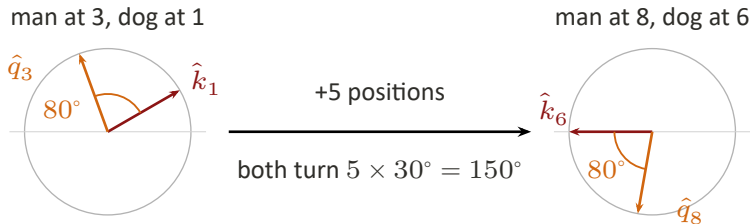
RoPE: Only the offset matters

Now move both words five positions later.



RoPE: Only the offset matters

Now move both words five positions later.



The angle between them still depends only on $t - j$.

$$\hat{q}_t^\top \hat{k}_j = (R_t q)^\top (R_j k) = q^\top R_t^\top R_j k = q^\top R_{j-t} k$$

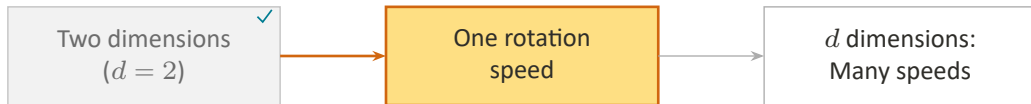
$$R_t^\top = R_{-t}, \text{ and } R_{-t} R_j = R_{j-t}.$$



Rotary position embedding (RoPE)

Su et al. (2021)

Rotate queries and keys by an angle proportional to position.



A rotation repeats. Can one speed tell near from far?

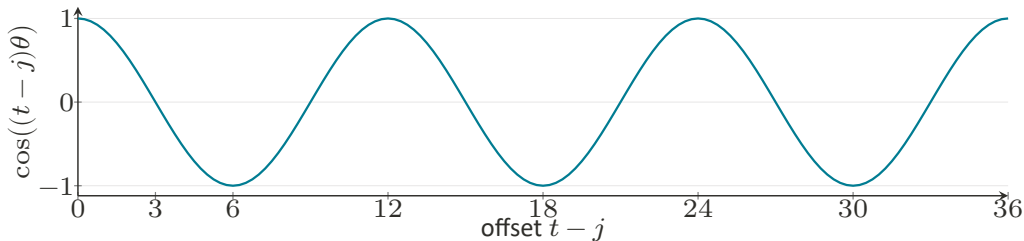


RoPE: Rotations repeat

In two dimensions: $\hat{q}_t^\top \hat{k}_j = \|q\| \|k\| \cos(\delta + (t - j)\theta)$.

Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).

30°/token: Period 12

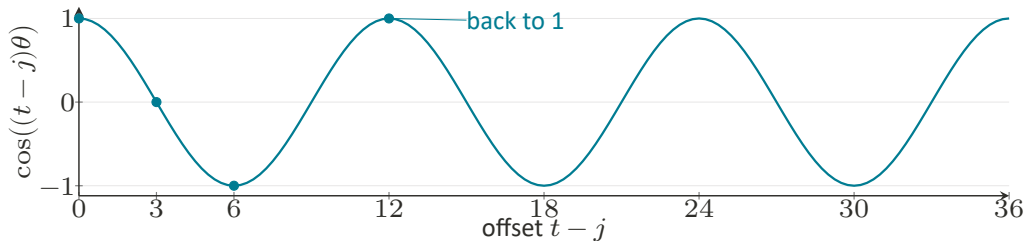


RoPE: Rotations repeat

In two dimensions: $\hat{q}_t^\top \hat{k}_j = \|q\| \|k\| \cos(\delta + (t - j)\theta)$.

Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).

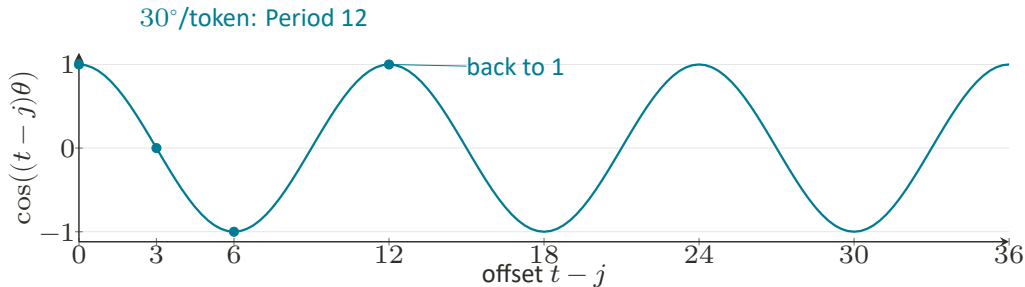
30°/token: Period 12



RoPE: Rotations repeat

In two dimensions: $\hat{q}_t^\top \hat{k}_j = \|q\| \|k\| \cos(\delta + (t - j)\theta)$.

Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).



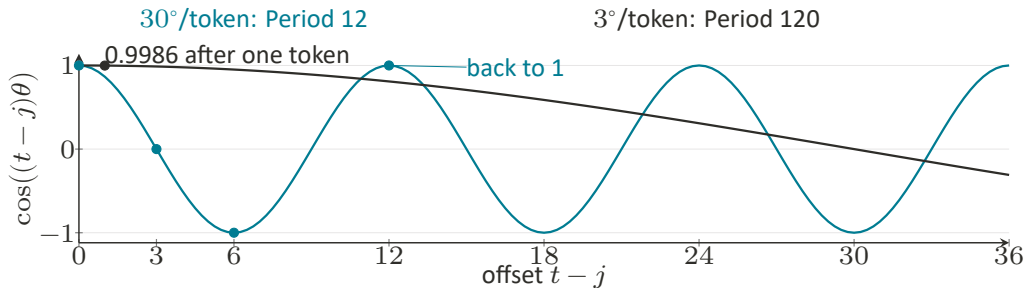
At 30° per token, offsets 0 and 12 get the same score.



RoPE: Fast versus slow speeds

At 3° per token, a full rotation takes 120 tokens.

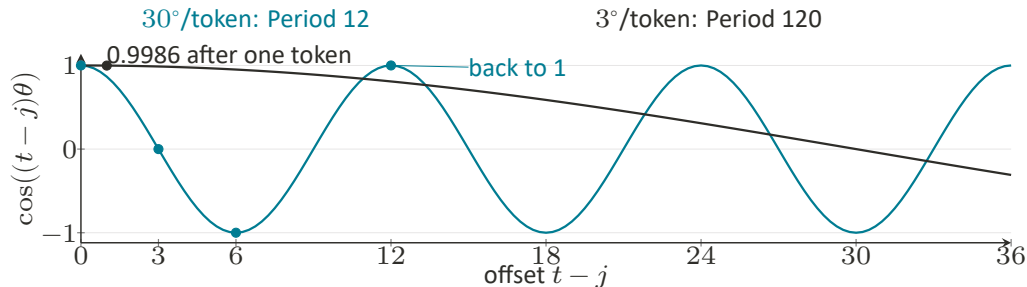
Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).



RoPE: Fast versus slow speeds

At 3° per token, a full rotation takes 120 tokens.

Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).



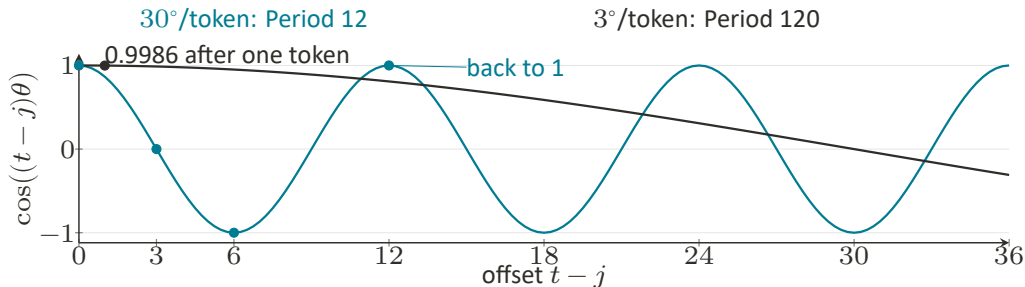
Offsets 0 and 1 score 1 and 0.9986, so neighbors look almost alike.



RoPE: Fast versus slow speeds

At 3° per token, a full rotation takes 120 tokens.

Plotted: $\cos((t - j)\theta)$, the score of unit vectors that start aligned ($\delta = 0$).



Offsets 0 and 1 score 1 and 0.9986, so neighbors look almost alike.

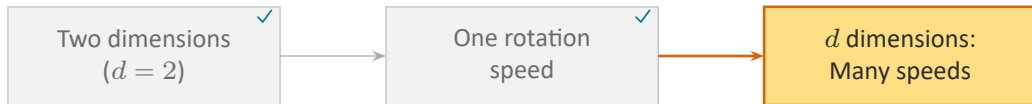
One speed struggles to distinguish offsets at both scales.



Rotary position embedding (RoPE)

Su et al. (2021)

Rotate queries and keys by an angle proportional to position.



Can different coordinates rotate at different speeds?



RoPE: Two pairs in four dimensions

Split $q \in \mathbb{R}^4$ into two pairs; each pair turns at its own speed.

$$\begin{array}{c} R(t\theta_1) \\ R(t\theta_2) \end{array} R_t \quad q \quad \hat{q}_t$$

	R_t		q	$\hat{q}_t$
$R(t\theta_1)$			q_1	$\hat{q}_1$
			q_2	$\hat{q}_2$
$R(t\theta_2)$	0	0	q_3	$\hat{q}_3$
	0	0	q_4	$\hat{q}_4$

=



RoPE: Two pairs in four dimensions

Split $q \in \mathbb{R}^4$ into two pairs; each pair turns at its own speed.

$$\begin{array}{c} R(t\theta_1) \\ R(t\theta_2) \end{array} R_t \quad q \quad \hat{q}_t$$

		0	0
		0	0
0	0		
0	0		

q_1
q_2
q_3
q_4

 $=$

$\hat{q}_1$
$\hat{q}_2$
$\hat{q}_3$
$\hat{q}_4$

 $\left\{ \begin{array}{l} \text{pair 1: } \theta_1 = 30^\circ \text{ per token} \\ \text{pair 2: } \theta_2 = 3^\circ \text{ per token} \end{array} \right.$

$$\hat{q}_t^\top \hat{k}_j = \underbrace{\hat{q}_{1:2}^\top \hat{k}_{1:2}}_{\text{pair 1}} + \underbrace{\hat{q}_{3:4}^\top \hat{k}_{3:4}}_{\text{pair 2}}$$

keys: Same pairs and speeds, angles $j\theta_1, j\theta_2$



RoPE: Two pairs in four dimensions

Split $q \in \mathbb{R}^4$ into two pairs; each pair turns at its own speed.

$$\begin{array}{c} R(t\theta_1) \\ R(t\theta_2) \end{array} R_t \quad q \quad \hat{q}_t$$

		0	0
		0	0
0	0		
0	0		

q_1
q_2
q_3
q_4

 $=$

$\hat{q}_1$
$\hat{q}_2$
$\hat{q}_3$
$\hat{q}_4$

 $\left\{ \begin{array}{l} \text{pair 1: } \theta_1 = 30^\circ \text{ per token} \\ \text{pair 2: } \theta_2 = 3^\circ \text{ per token} \end{array} \right.$

$$\hat{q}_t^\top \hat{k}_j = \underbrace{\hat{q}_{1:2}^\top \hat{k}_{1:2}}_{\text{pair 1}} + \underbrace{\hat{q}_{3:4}^\top \hat{k}_{3:4}}_{\text{pair 2}}$$

keys: Same pairs and speeds, angles $j\theta_1, j\theta_2$

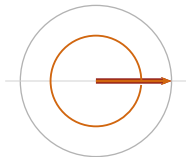
Each pair follows the two-dimensional case, at its own speed.



RoPE: Two speeds at offset 12

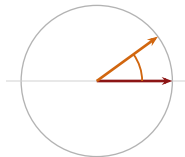
For aligned unit pairs at offset 12, add the two pair scores.

pair 1: 30° per token



gap 360° : Score 1

pair 2: 3° per token



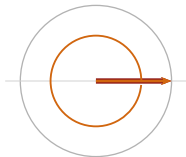
gap 36° : Score 0.81



RoPE: Two speeds at offset 12

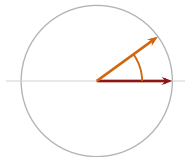
For aligned unit pairs at offset 12, add the two pair scores.

pair 1: 30° per token



gap 360° : Score 1

pair 2: 3° per token



gap 36° : Score 0.81

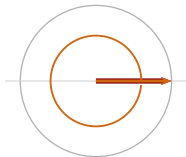
Offset 12 scores $1 + 0.81$; offset 0 scores $1 + 1$.



RoPE: Two speeds at offset 12

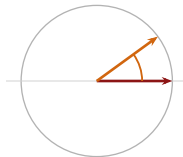
For aligned unit pairs at offset 12, add the two pair scores.

pair 1: 30° per token



gap 360° : Score 1

pair 2: 3° per token



gap 36° : Score 0.81

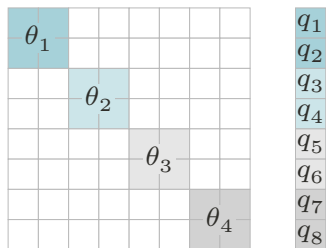
Offset 12 scores $1 + 0.81$; offset 0 scores $1 + 1$.

Fast pairs separate nearby offsets; slow pairs separate far ones.



RoPE in d dimensions

Split $q, k \in \mathbb{R}^d$ into $d/2$ pairs, each with its own speed:

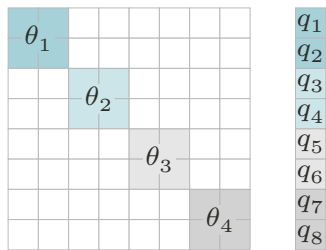


$d = 8$ shown; pair i = coordinates $2i - 1, 2i$



RoPE in d dimensions

Split $q, k \in \mathbb{R}^d$ into $d/2$ pairs, each with its own speed:



$$\theta_i = 10000^{-2(i-1)/d}, \quad i = 1, \dots, d/2$$

Pair 1 turns once every 6.3 tokens.

At $d = 128$, pair 64 turns every 54,000.

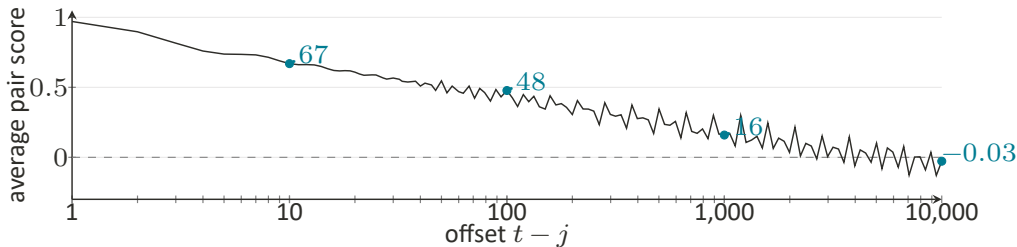
$d = 8$ shown; pair i = coordinates $2i - 1, 2i$



RoPE: Many speeds together

Plotted: The score of aligned unit vectors with $d = 128$,

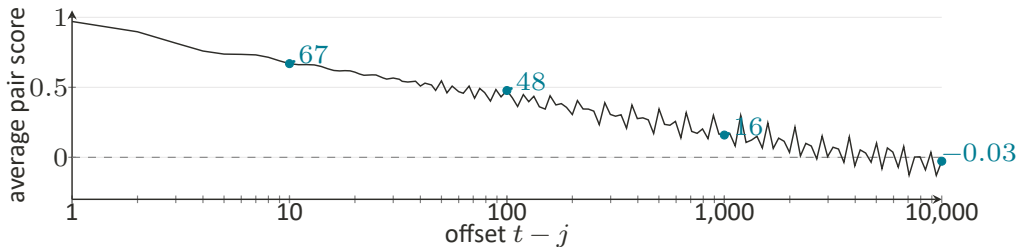
$$\frac{1}{64} \sum_{i=1}^{64} \cos((t - j) \theta_i), \quad \text{with } \theta_i = 10000^{-(i-1)/64}.$$



RoPE: Many speeds together

Plotted: The score of aligned unit vectors with $d = 128$,

$$\frac{1}{64} \sum_{i=1}^{64} \cos((t - j) \theta_i), \quad \text{with } \theta_i = 10000^{-(i-1)/64}.$$



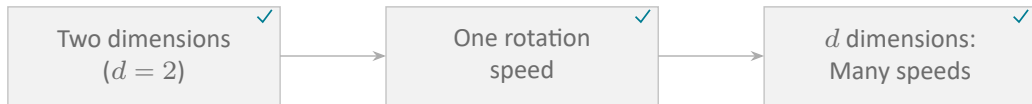
The speeds agree only near offset 0, so this score tends to fall.



Rotary position embedding (RoPE)

Su et al. (2021)

Rotate queries and keys by an angle proportional to position.



RoPE: The score depends on $t - j$, at many scales.



How recent models use RoPE

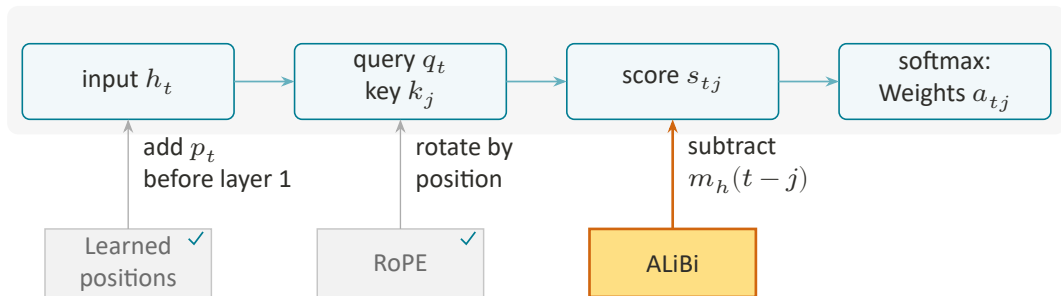
Change	Example	Setting
Different base by layer	Gemma 3 27B	base 10K local; 1M global ↑ local layers: Sliding window
Rotate only some pairs	GLM-4.5	32 of 64 pairs
Some layers skip it	Llama 4 Scout	every fourth layer

$$\theta_i = \text{base}^{-2(i-1)/d}$$

Source: [Model reports and configurations](#).



Positional encodings



Can we build a preference for recent tokens into every head?



Subtract a distance penalty before softmax.

$$\tilde{s}_{tj} = s_{tj} - m_h(t - j), \quad a_{tj} = \text{softmax}_j(\tilde{s}_{tj})$$

attention logit: $q_t^\top k_j / \sqrt{d_k}$

slope of head h : A fixed hyperparameter

Source: Press et al. (2021), Figure 3. Queries, keys, and values stay unchanged.



ALiBi

Press et al. (2021)

Subtract a distance penalty before softmax.

$$\tilde{s}_{tj} = s_{tj} - m_h(t - j), \quad a_{tj} = \text{softmax}_j(\tilde{s}_{tj})$$

attention logit: $q_t^\top k_j / \sqrt{d_k}$

slope of head h : A fixed hyperparameter

The diagram illustrates the ALiBi modification to attention logits. It shows a 5x5 matrix of attention logits (left) and a 5x5 matrix of distance penalties (right), which are added together and scaled by a slope m .

Attention Logit Matrix (Left):

$q_1 \cdot k_1$				
$q_2 \cdot k_1$	$q_2 \cdot k_2$			
$q_3 \cdot k_1$	$q_3 \cdot k_2$	$q_3 \cdot k_3$		
$q_4 \cdot k_1$	$q_4 \cdot k_2$	$q_4 \cdot k_3$	$q_4 \cdot k_4$	
$q_5 \cdot k_1$	$q_5 \cdot k_2$	$q_5 \cdot k_3$	$q_5 \cdot k_4$	$q_5 \cdot k_5$

Distance Penalty Matrix (Right):

0				
-1	0			
-2	-1	0		
-3	-2	-1	0	
-4	-3	-2	-1	0

The operation is: Attention Logit Matrix + Distance Penalty Matrix $\cdot m$.

Source: Press et al. (2021), Figure 3. Queries, keys, and values stay unchanged.



Subtract a distance penalty before softmax.

$$\tilde{s}_{tj} = s_{tj} - m_h(t - j), \quad a_{tj} = \text{softmax}_j(\tilde{s}_{tj})$$

attention logit: $q_t^\top k_j / \sqrt{d_k}$

slope of head h : A fixed hyperparameter

$$\begin{bmatrix} q_1 \cdot k_1 & & & & \\ q_2 \cdot k_1 & q_2 \cdot k_2 & & & \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & & \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 & \\ q_5 \cdot k_1 & q_5 \cdot k_2 & q_5 \cdot k_3 & q_5 \cdot k_4 & q_5 \cdot k_5 \end{bmatrix} + \begin{bmatrix} 0 & & & & \\ -1 & 0 & & & \\ -2 & -1 & 0 & & \\ -3 & -2 & -1 & 0 & \\ -4 & -3 & -2 & -1 & 0 \end{bmatrix} \cdot m$$

Eight heads: Slopes $1/2, 1/4, \dots, 1/256$.

Source: Press et al. (2021), Figure 3. Queries, keys, and values stay unchanged.



ALiBi: Example

Query: man. Equal logits s ; slope $m = \log 2$.

dog

distance 2

bites

distance 1

man

distance 0



ALiBi: Example

Query: man. Equal logits s ; slope $m = \log 2$.

	dog	bites	man
	distance 2	distance 1	distance 0
subtract the distance penalty	$s - 2 \log 2$	$s - \log 2$	s



ALiBi: Example

Query: man. Equal logits s ; slope $m = \log 2$.

	dog distance 2	bites distance 1	man distance 0
subtract the distance penalty	$s - 2 \log 2$	$s - \log 2$	s
	↓	↓	↓
exponentiate: $\propto$	$1/4$	$1/2$	1



ALiBi: Example

Query: man. Equal logits s ; slope $m = \log 2$.

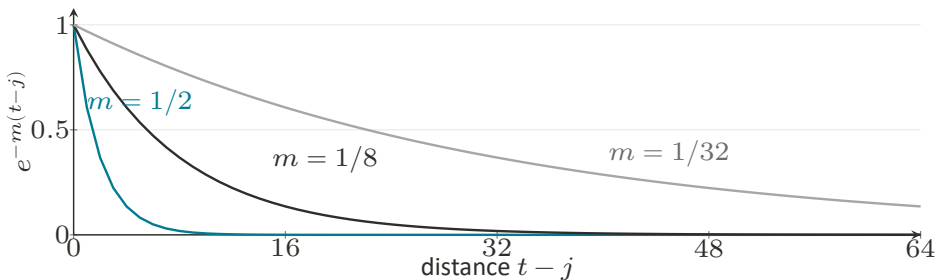
	dog distance 2	bites distance 1	man distance 0
subtract the distance penalty	$s - 2 \log 2$	$s - \log 2$	s
	↓	↓	↓
exponentiate: $\propto$	$1/4$	$1/2$	1
			$4/7$
normalize	$1/7$	$2/7$	

The unnormalized weight decays exponentially with distance.



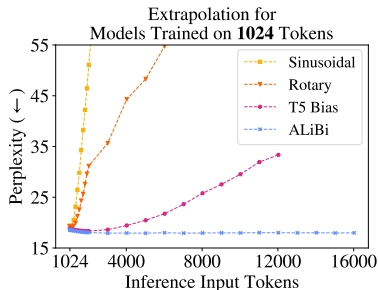
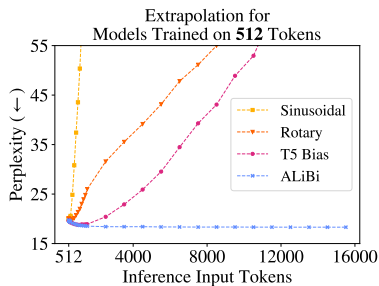
ALiBi: Different memory spans

Each head's slope sets how quickly attention decays with distance.



ALiBi and RoPE on longer inputs

247M parameters, WikiText-103; trained at 512 or 1,024 tokens.

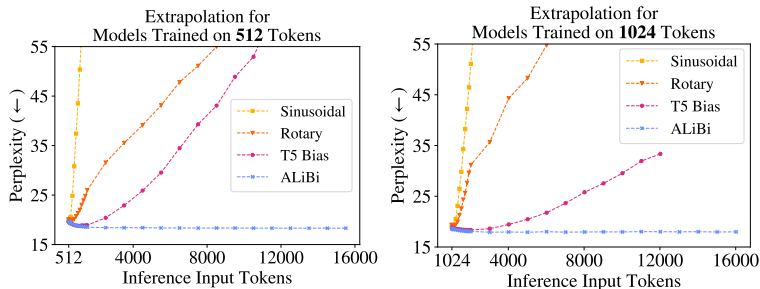


Source: [Press et al. \(2021\)](#), extrapolation experiments.



ALiBi and RoPE on longer inputs

247M parameters, WikiText-103; trained at 512 or 1,024 tokens.

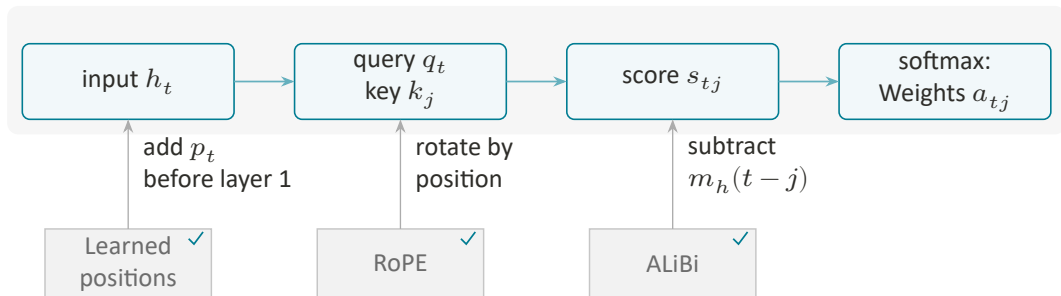


ALiBi's perplexity stays nearly flat out to 16K; RoPE's rises.

Source: [Press et al. \(2021\)](#), extrapolation experiments.



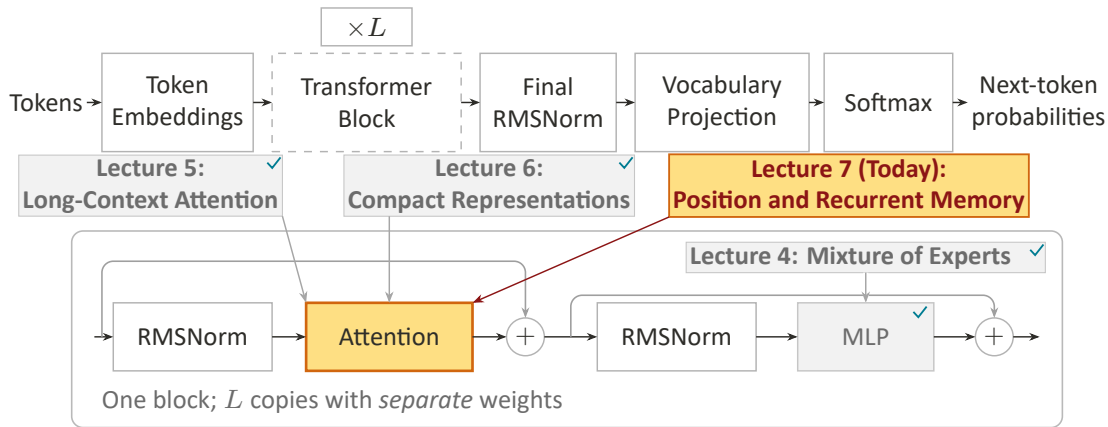
Positional encodings



All three still keep a key and value for every token.



Topic 2: Efficient attention

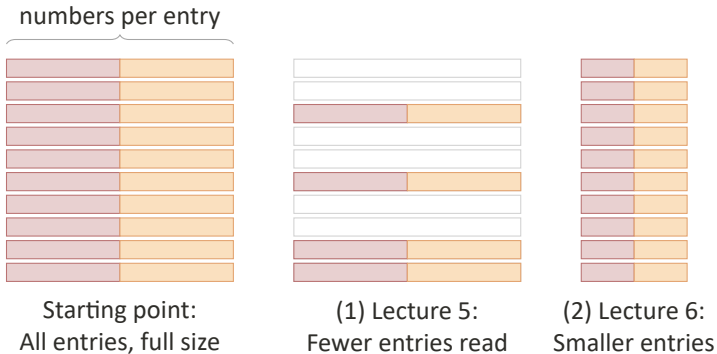


Can we reduce the cost per token while keeping quality fixed?



Different types of caches for inference

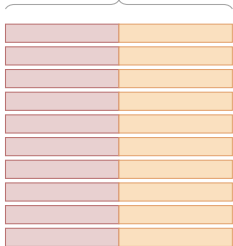
Every layer keeps information from earlier tokens.



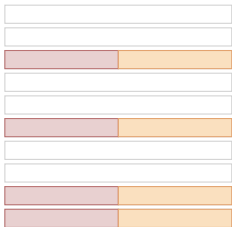
Different types of caches for inference

Every layer keeps information from earlier tokens.

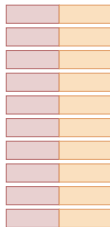
numbers per entry



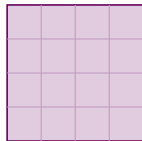
Starting point:
All entries, full size



(1) Lecture 5:
Fewer entries read



(2) Lecture 6:
Smaller entries

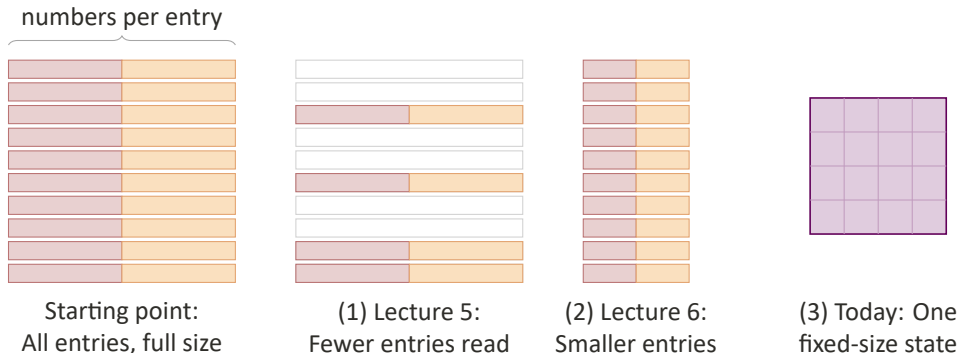


(3) Today: One
fixed-size state



Different types of caches for inference

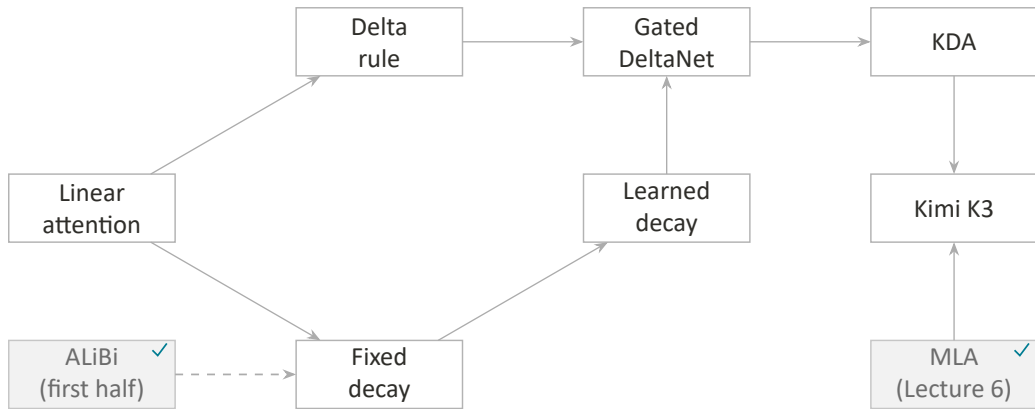
Every layer keeps information from earlier tokens.



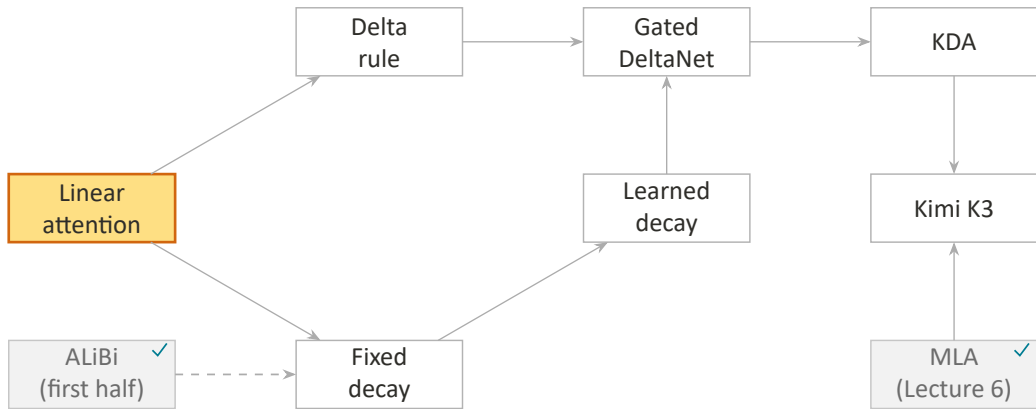
Lectures 5 and 6 shrink the KV cache, but it still grows with T .



Innovations in recurrent memory



Innovations in recurrent memory



Linear attention

With softmax, attention on $Q, K, V \in \mathbb{R}^{T \times d}$ takes $O(T^2 d)$ time:

$$\text{softmax}(\underbrace{QK^T}_{T \times T}) V$$



Linear attention

With softmax, attention on $Q, K, V \in \mathbb{R}^{T \times d}$ takes $O(T^2 d)$ time:

$$\text{softmax}(\underbrace{QK^T}_{T \times T}) V$$

Without softmax, we can regroup the product:

$$\underbrace{(QK^T)}_{T \times T} V = Q \underbrace{(K^T V)}_{d \times d}$$



Linear attention

With softmax, attention on $Q, K, V \in \mathbb{R}^{T \times d}$ takes $O(T^2 d)$ time:

$$\text{softmax}(\underbrace{QK^T}_{T \times T}) V$$

Without softmax, we can regroup the product:

$$\underbrace{(QK^T)}_{T \times T} V = Q \underbrace{(K^T V)}_{d \times d}$$

The right-hand side takes $O(Td^2)$ time: Faster when $T \gg d$.

How can we keep softmax's useful properties without it?



Desirable properties of softmax

Softmax weights are nonnegative and sum to one:

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_i e^{s_{ti}}} \geq 0, \quad \sum_j a_{tj} = 1$$



Desirable properties of softmax

Softmax weights are nonnegative and sum to one:

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_i e^{s_{ti}}} \geq 0, \quad \sum_j a_{tj} = 1$$

So the output $o_t = \sum_j a_{tj} v_j$ is a weighted average of the values:

$$\min_j v_{jc} \leq o_{tc} \leq \max_j v_{jc} \quad \text{for each coordinate } c$$



Desirable properties of softmax

Softmax weights are nonnegative and sum to one:

$$a_{tj} = \frac{e^{s_{tj}}}{\sum_i e^{s_{ti}}} \geq 0, \quad \sum_j a_{tj} = 1$$

So the output $o_t = \sum_j a_{tj} v_j$ is a weighted average of the values:

$$\min_j v_{jc} \leq o_{tc} \leq \max_j v_{jc} \quad \text{for each coordinate } c$$

This ensures that o_t cannot grow with the context length T , and that changing one value by ε moves o_t by at most $|\varepsilon|$.



Linear attention: Normalization

Katharopoulos et al. (2020)

To keep the weights summing to one, divide each row by its sum:

$$o_t = \frac{\sum_j (q_t^\top k_j) v_j}{\sum_j q_t^\top k_j}$$



Linear attention: Normalization

Katharopoulos et al. (2020)

To keep the weights summing to one, divide each row by its sum:

$$o_t = \frac{\sum_j (q_t^\top k_j) v_j}{\sum_j q_t^\top k_j} = \frac{(V^\top K) q_t}{(K^\top \mathbf{1})^\top q_t}$$

$\mathbf{1}$: A column of T ones



Linear attention: Normalization

Katharopoulos et al. (2020)

To keep the weights summing to one, divide each row by its sum:

$$o_t = \frac{\sum_j (q_t^\top k_j) v_j}{\sum_j q_t^\top k_j} = \frac{(V^\top K) q_t}{(K^\top \mathbf{1})^\top q_t}$$

$\mathbf{1}$: A column of T ones

The denominator regroups the same way, adding only $O(Td)$ time.



Normalizing does not bound the output

Normalized weights sum to one, but each $q_t^\top k_j$ can be negative.



Normalizing does not bound the output

Normalized weights sum to one, but each $q_t^\top k_j$ can be negative.

Example: Weights 100 and -99 sum to one, and $100(5) - 99(5) = 5$.



Normalizing does not bound the output

Normalized weights sum to one, but each $q_t^\top k_j$ can be negative.

Example: Weights 100 and -99 sum to one, and $100(5) - 99(5) = 5$.

Issue: Changing the first value to 5.01 makes the output jump to 6.



Normalizing does not bound the output

Normalized weights sum to one, but each $q_t^\top k_j$ can be negative.

Example: Weights 100 and -99 sum to one, and $100(5) - 99(5) = 5$.

Issue: Changing the first value to 5.01 makes the output jump to 6.

One way to bound the output is to make every weight nonnegative.

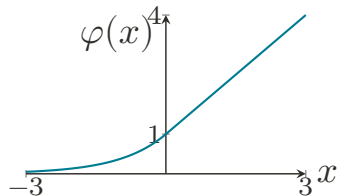


Linear attention: Positive features

Katharopoulos et al. (2020)

Apply a positive function to each coordinate of q_t and k_j :

$$\varphi(x) = \text{ELU}(x) + 1 > 0$$

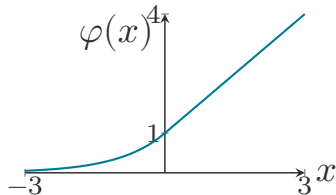


Linear attention: Positive features

Katharopoulos et al. (2020)

Apply a positive function to each coordinate of q_t and k_j :

$$\varphi(x) = \text{ELU}(x) + 1 > 0$$



Now every dot product $\varphi(q_t)^\top \varphi(k_j)$ is positive:

$$o_t = \frac{\sum_j \varphi(q_t)^\top \varphi(k_j) v_j}{\sum_j \varphi(q_t)^\top \varphi(k_j)}$$

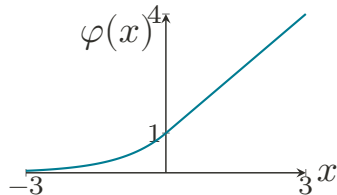


Linear attention: Positive features

Katharopoulos et al. (2020)

Apply a positive function to each coordinate of q_t and k_j :

$$\varphi(x) = \text{ELU}(x) + 1 > 0$$



Now every dot product $\varphi(q_t)^\top \varphi(k_j)$ is positive:

$$o_t = \frac{\sum_j \varphi(q_t)^\top \varphi(k_j) v_j}{\sum_j \varphi(q_t)^\top \varphi(k_j)}$$

The weights are nonnegative and sum to one, as with softmax.



Linear attention: Recurrent form

Katharopoulos et al. (2020)

Linear attention can be viewed as a recurrence:

$$S_t = S_{t-1} + v_t \varphi(k_t)^\top$$

$$z_t = z_{t-1} + \varphi(k_t)$$

$$o_t = \frac{S_t \varphi(q_t)}{z_t^\top \varphi(q_t)}$$



Linear attention: Recurrent form

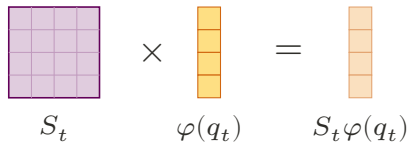
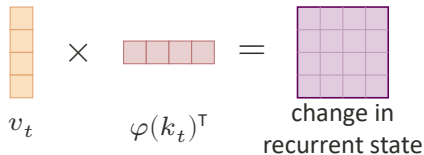
Katharopoulos et al. (2020)

Linear attention can be viewed as a recurrence:

$$S_t = S_{t-1} + v_t \varphi(k_t)^\top$$

$$z_t = z_{t-1} + \varphi(k_t)$$

$$o_t = \frac{S_t \varphi(q_t)}{z_t^\top \varphi(q_t)}$$



Linear attention: Recurrent form

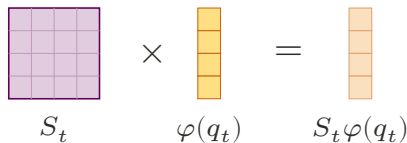
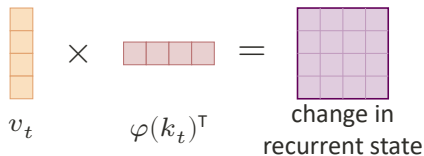
Katharopoulos et al. (2020)

Linear attention can be viewed as a recurrence:

$$S_t = S_{t-1} + v_t \varphi(k_t)^\top$$

$$z_t = z_{t-1} + \varphi(k_t)$$

$$o_t = \frac{S_t \varphi(q_t)}{z_t^\top \varphi(q_t)}$$



We only need to store S_t and z_t , and their size does not grow with the context.



Linear attention: Recurrent form

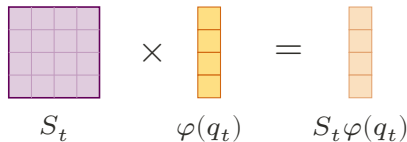
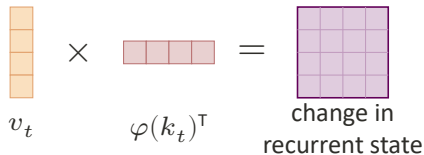
Katharopoulos et al. (2020)

Linear attention can be viewed as a recurrence:

$$S_t = S_{t-1} + v_t \varphi(k_t)^\top \longleftarrow S_t = \sum_{j \leq t} v_j \varphi(k_j)^\top$$

$$z_t = z_{t-1} + \varphi(k_t)$$

$$o_t = \frac{S_t \varphi(q_t)}{z_t^\top \varphi(q_t)} \longleftarrow o_t = \frac{\sum_{j \leq t} \varphi(q_t)^\top \varphi(k_j) v_j}{\sum_{j \leq t} \varphi(q_t)^\top \varphi(k_j)}$$



We only need to store S_t and z_t , and their size does not grow with the context.



Limitations of a fixed-size state

In addition to earlier issues, maintaining a fixed-size state can also:



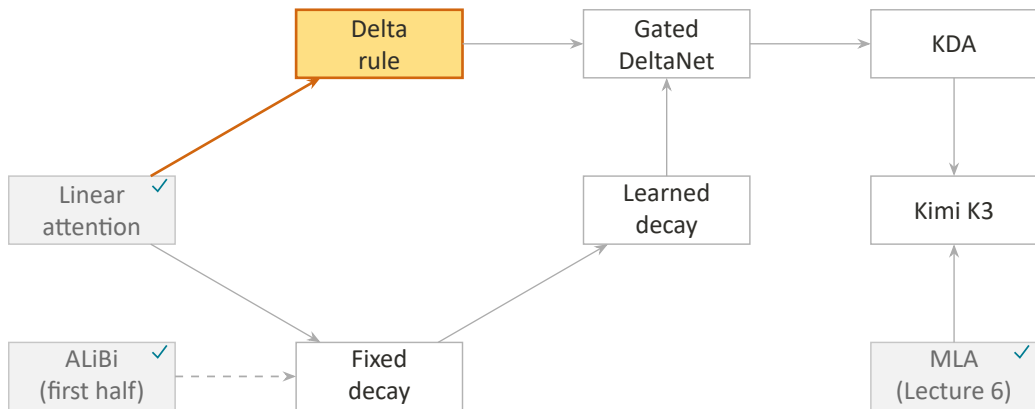
Limitations of a fixed-size state

In addition to earlier issues, maintaining a fixed-size state can also:

1. make larger changes than necessary (we will see why);
2. give updates from long ago as much weight as new ones;
3. lose information from earlier tokens.



Innovations in recurrent memory



Can a write replace an outdated value?



Issue 1: Larger changes than necessary

Stylized example

Keys are $\text{apple} = (1, 0)$ and $\text{bread} = (0, 1)$, and values are prices.

apple 2

bread 3

apple 5

apple bread

2

3



Issue 1: Larger changes than necessary

Stylized example

Keys are apple = $(1, 0)$ and bread = $(0, 1)$, and values are prices.

apple 2

bread 3

apple 5

apple	bread
2	3

add $5k^T$

apple	bread
7	3

apple: 7; normalized: 3.5

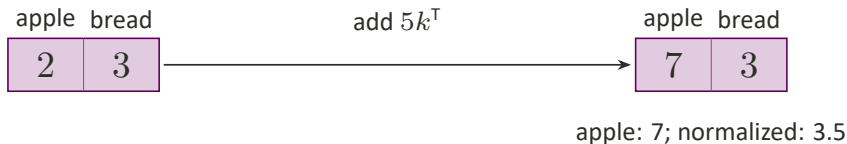


Issue 1: Larger changes than necessary

Stylized example

Keys are apple = (1, 0) and bread = (0, 1), and values are prices.

apple 2 bread 3 apple 5



We want the read to return the current price, 5.

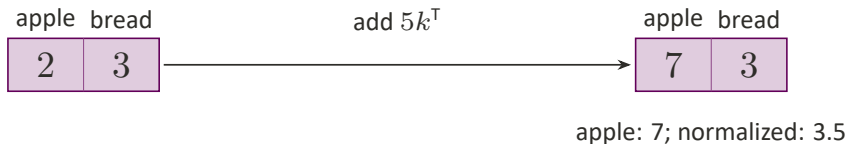


Issue 1: Larger changes than necessary

Stylized example

Keys are apple = (1, 0) and bread = (0, 1), and values are prices.

apple 2 bread 3 apple 5



We want the read to return the current price, 5.

Adding the full 5 changes the state more than needed: A change of $5 - 2 = 3$ would suffice.



Idea 1: First-principles update

Widrow and Hoff (1960)

Treat S as a linear predictor from keys to values, with loss

$$\ell_t(S) = \frac{1}{2} \|Sk_t - v_t\|^2$$



Idea 1: First-principles update

Widrow and Hoff (1960)

Treat S as a linear predictor from keys to values, with loss

$$\ell_t(S) = \frac{1}{2} \|Sk_t - v_t\|^2$$

Its gradient is $\nabla_S \ell_t(S) = (Sk_t - v_t) k_t^\top$.



Idea 1: First-principles update

Widrow and Hoff (1960)

Treat S as a linear predictor from keys to values, with loss

$$\ell_t(S) = \frac{1}{2} \|Sk_t - v_t\|^2$$

Its gradient is $\nabla_S \ell_t(S) = (Sk_t - v_t) k_t^\top$.

Performing gradient descent with step size β_t gives the update rule:

$$S_t = S_{t-1} - \beta_t \nabla_S \ell_t(S_{t-1}) = S_{t-1} + \beta_t (v_t - \hat{v}_t) k_t^\top$$

$\hat{v}_t = S_{t-1} k_t$, the current prediction



Delta rule

Widrow and Hoff (1960); Schlag et al. (2021)

Using this gradient step to update the linear-attention state gives

$$\hat{v}_t = S_{t-1} k_t$$

$$S_t = S_{t-1} + \beta_t (v_t - \hat{v}_t) k_t^\top$$

$$o_t = S_t q_t$$



Delta rule

Widrow and Hoff (1960); Schlag et al. (2021)

Using this gradient step to update the linear-attention state gives

$$\hat{v}_t = S_{t-1} k_t$$

$$S_t = S_{t-1} + \beta_t (v_t - \hat{v}_t) k_t^\top$$

$$o_t = S_t q_t$$

Like q_t, k_t, v_t , the step size $\beta_t \in [0, 1]$ is computed from h_t .



Delta rule

Widrow and Hoff (1960); Schlag et al. (2021)

Using this gradient step to update the linear-attention state gives

$$\hat{v}_t = S_{t-1} k_t$$

$$S_t = S_{t-1} + \beta_t (v_t - \hat{v}_t) k_t^\top$$

$$o_t = S_t q_t$$

Like q_t, k_t, v_t , the step size $\beta_t \in [0, 1]$ is computed from h_t .

The output $o_t = S_t q_t$ needs no division by a sum of weights.

$$\|k_t\| = 1 : \quad S_t k_t = (1 - \beta_t) \hat{v}_t + \beta_t v_t$$

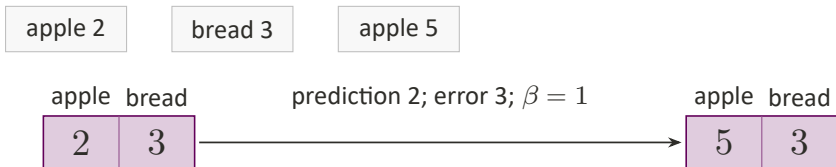
Each write moves the prediction toward v_t but does not “overshoot”.



Delta rule: Example

Stylized example

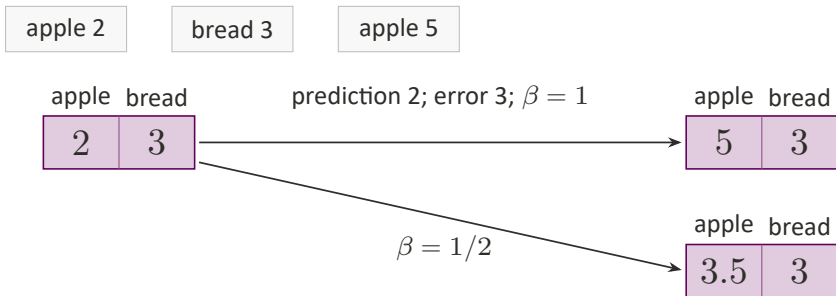
Start from $S = [2 \ 3]$ and write apple's new price, 5, with key $(1, 0)$.



Delta rule: Example

Stylized example

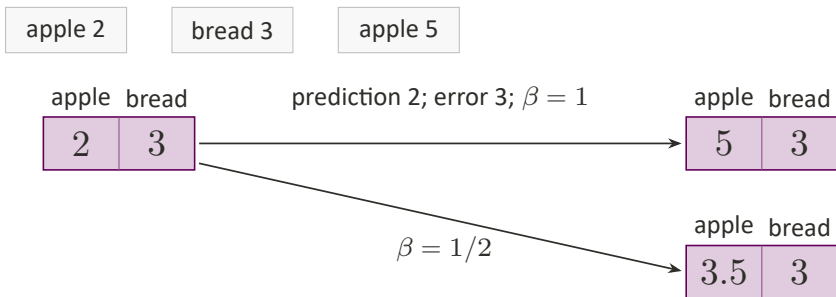
Start from $S = [2 \ 3]$ and write apple's new price, 5, with key $(1, 0)$.



Delta rule: Example

Stylized example

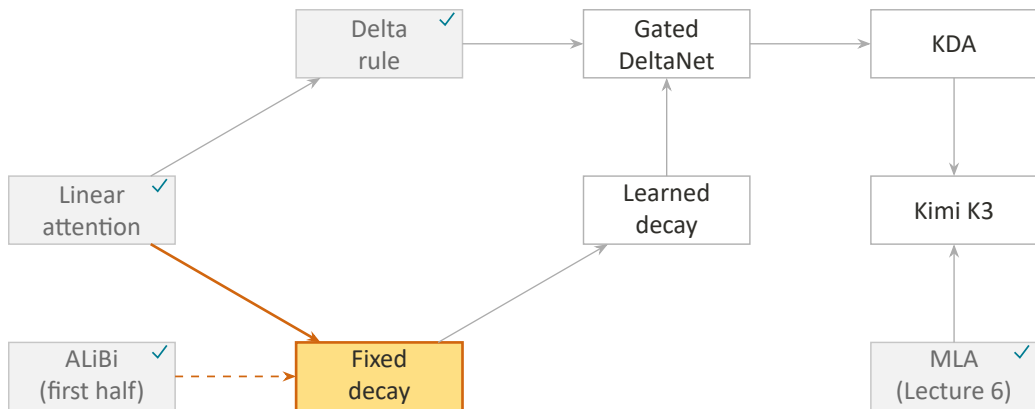
Start from $S = [2 \ 3]$ and write apple's new price, 5, with key $(1, 0)$.



Writing apple 5 again changes nothing because the prediction is 5.
Bread's price does not change, as its key is orthogonal to apple's.



Innovations in recurrent memory



Can older writes count less than recent ones?



Fixed decay

Sun et al. (2023)

Older writes are often less relevant, so they should count less.

With a KV cache, positional encodings give distant tokens less weight.

A recurrent state cannot do this, as added writes lose their positions.



Fixed decay

Sun et al. (2023)

Older writes are often less relevant, so they should count less.

With a KV cache, positional encodings give distant tokens less weight.

A recurrent state cannot do this, as added writes lose their positions.

Idea: Decay the state by a fixed $\gamma \in (0, 1)$ before each write:

$$S_t = \gamma S_{t-1} + v_t k_t^\top = \sum_{j \leq t} \gamma^{t-j} v_j k_j^\top$$



Fixed decay

Sun et al. (2023)

Older writes are often less relevant, so they should count less.

With a KV cache, positional encodings give distant tokens less weight.

A recurrent state cannot do this, as added writes lose their positions.

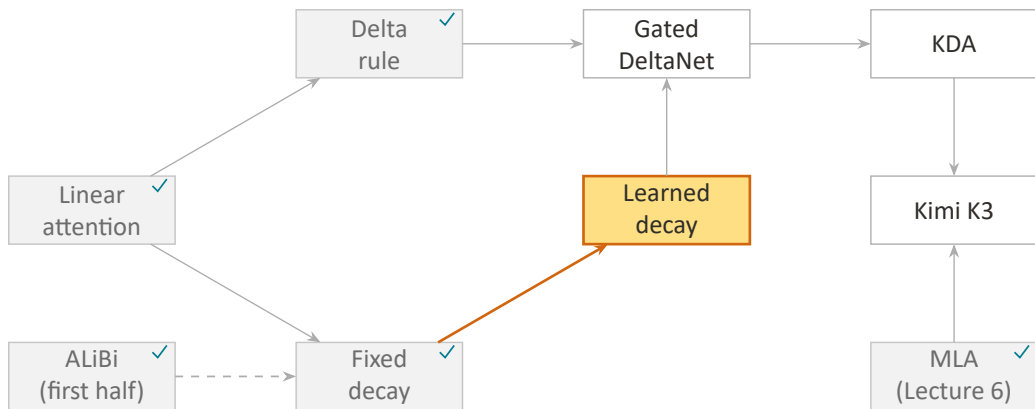
Idea: Decay the state by a fixed $\gamma \in (0, 1)$ before each write:

$$S_t = \gamma S_{t-1} + v_t k_t^\top = \sum_{j \leq t} \gamma^{t-j} v_j k_j^\top$$

A write made a tokens ago now has weight γ^a .



Innovations in recurrent memory



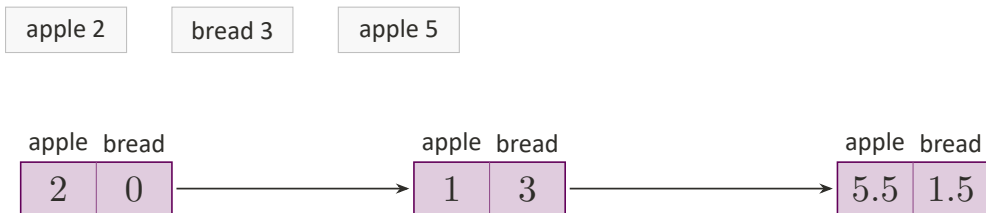
Should every write decay at the same fixed rate?



Fixed decay: Example

Stylized example

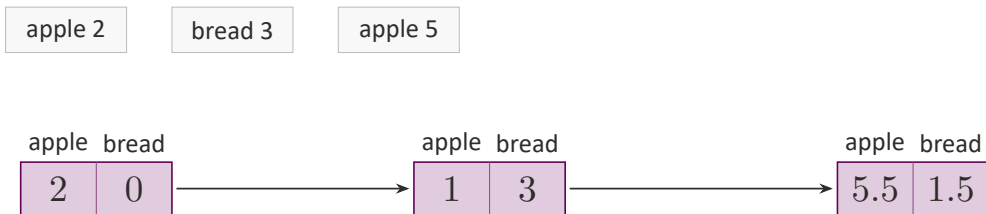
Return to the price example, now with decay $\gamma = 1/2$.



Fixed decay: Example

Stylized example

Return to the price example, now with decay $\gamma = 1/2$.



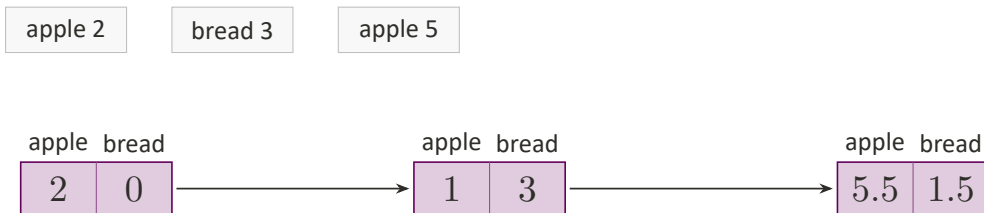
The price of bread decays even though it never changed.



Fixed decay: Example

Stylized example

Return to the price example, now with decay $\gamma = 1/2$.



The price of bread decays even though it never changed.

Can the model learn which writes to decay?



Learned decay

Yang et al. (2023)

Let the input decide how much each feature decays:

$$S_t = S_{t-1} \text{Diag}(\alpha_t) + v_t k_t^T$$

The input h_t sets $\alpha_t \in (0, 1)^{d_k}$, one decay factor per column of S .



Learned decay

Yang et al. (2023)

Let the input decide how much each feature decays:

$$S_t = S_{t-1} \text{Diag}(\alpha_t) + v_t k_t^T$$

The input h_t sets $\alpha_t \in (0, 1)^{d_k}$, one decay factor per column of S .

Fixed decay is the special case $\alpha_t = (\gamma, \dots, \gamma)$.



Learned decay

Yang et al. (2023)

Let the input decide how much each feature decays:

$$S_t = S_{t-1} \text{Diag}(\alpha_t) + v_t k_t^T$$

The input h_t sets $\alpha_t \in (0, 1)^{d_k}$, one decay factor per column of S .

Fixed decay is the special case $\alpha_t = (\gamma, \dots, \gamma)$.

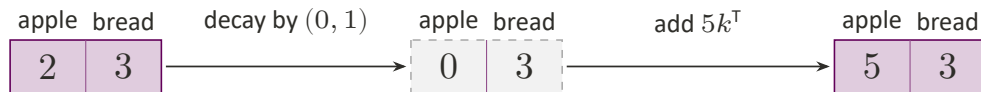
Each “feature” can now keep or erase its writes separately.



Learned decay: Example

Stylized example

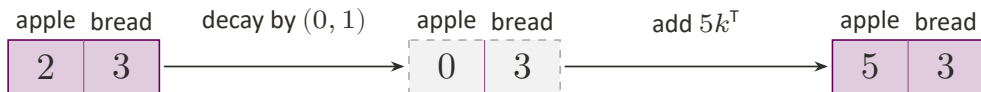
Suppose $\alpha = (0, 1)$ and a new price of apple arrives. Then,



Learned decay: Example

Stylized example

Suppose $\alpha = (0, 1)$ and a new price of apple arrives. Then,



Learned decay gives the current prices, $[5 \ 3]$.



Gated DeltaNet

Yang et al. (2024)

Gated DeltaNet combines both ideas:
Learned decay and the delta rule.

$$\tilde{S}_t = \alpha_t S_{t-1}, \quad S_t = \tilde{S}_t + \beta_t (v_t - \tilde{S}_t k_t) k_t^\top$$



Gated DeltaNet

Yang et al. (2024)

Gated DeltaNet combines both ideas:
Learned decay and the delta rule.

$$\tilde{S}_t = \alpha_t S_{t-1}, \quad S_t = \tilde{S}_t + \beta_t (v_t - \tilde{S}_t k_t) k_t^\top$$

gate (learned decay) delta-rule correction



Gated DeltaNet

Yang et al. (2024)

Gated DeltaNet combines both ideas:
Learned decay and the delta rule.

$$\tilde{S}_t = \alpha_t S_{t-1}, \quad S_t = \tilde{S}_t + \beta_t (v_t - \tilde{S}_t k_t) k_t^\top$$

gate (learned decay) delta-rule correction

Gated DeltaNet uses a scalar gate, one $\alpha_t \in (0, 1)$ per head, computed from h_t .

With one gate per head, every feature decays by the same factor.



Kimi Delta Attention (KDA) Kimi Team (2025)

KDA uses learned decay per feature, then corrects with the delta rule:

$$\tilde{S} = S_{t-1} \text{Diag}(\alpha_t), \quad S_t = \tilde{S} + \beta_t (v_t - \tilde{S} k_t) k_t^\top$$

This allows more general decay, with one factor per feature instead of one per head.



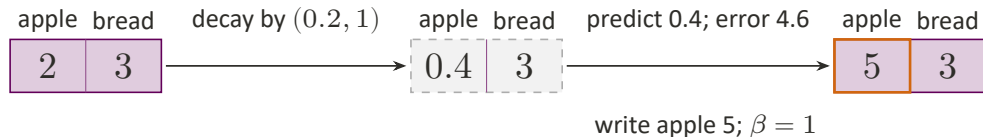
Kimi Delta Attention (KDA) Kimi Team (2025)

KDA uses learned decay per feature, then corrects with the delta rule:

$$\tilde{S} = S_{t-1} \text{Diag}(\alpha_t), \quad S_t = \tilde{S} + \beta_t(v_t - \tilde{S}k_t)k_t^\top$$

This allows more general decay, with one factor per feature instead of one per head.

Stylized example



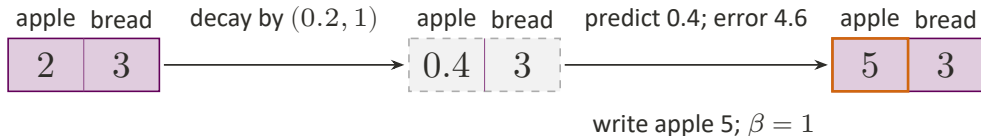
Kimi Delta Attention (KDA) Kimi Team (2025)

KDA uses learned decay per feature, then corrects with the delta rule:

$$\tilde{S} = S_{t-1} \text{Diag}(\alpha_t), \quad S_t = \tilde{S} + \beta_t(v_t - \tilde{S}k_t)k_t^\top$$

This allows more general decay, with one factor per feature instead of one per head.

Stylized example



Apple now reads 5, and bread keeps its price of 3.



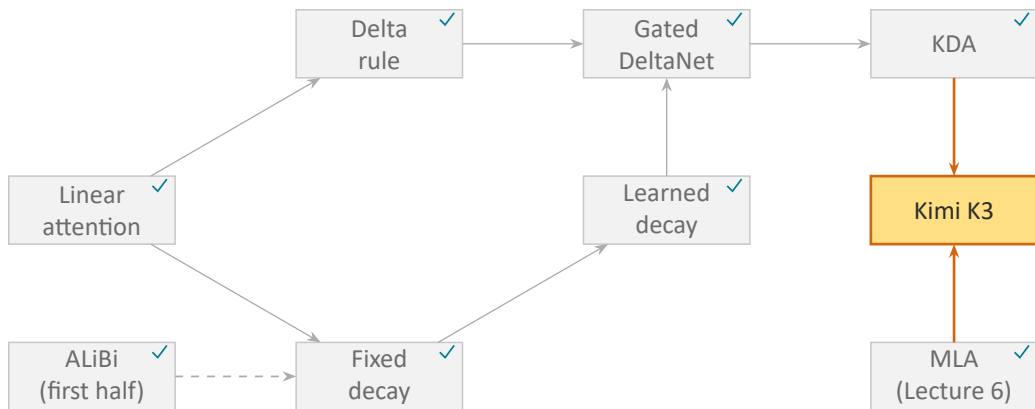
Design choices for recurrent memory

Several choices of decay rules and write rules:

	Add vk^T	Correct the error
No decay	Linear attention (2020)	Delta rule (2021)
Fixed decay	RetNet (2023)	
Learned decay, per head	Mamba-2 (2024)	Gated DeltaNet (2024)
Learned decay, per feature	GLA (2023)	KDA (2025)



Innovations in recurrent memory



Can some layers still store every token's key and value?



Kimi K3: KDA and MLA layers

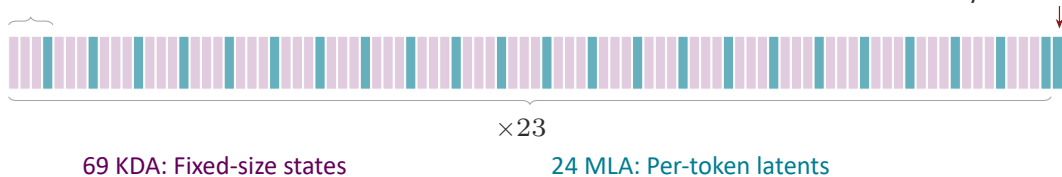
Kimi Team (2026)

K3 has 93 layers: 23 blocks of (KDA, KDA, KDA, MLA),
then a final MLA layer.

It has 2.8T parameters, of which 104B are active for each token.

3 KDA + 1 MLA

final MLA: Layer 93



Kimi K3: KDA and MLA layers

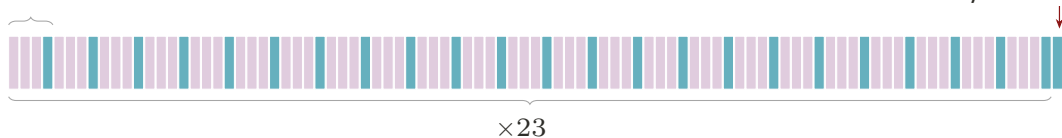
Kimi Team (2026)

K3 has 93 layers: 23 blocks of (KDA, KDA, KDA, MLA),
then a final MLA layer.

It has 2.8T parameters, of which 104B are active for each token.

3 KDA + 1 MLA

final MLA: Layer 93



69 KDA: Fixed-size states

24 MLA: Per-token latents

KDA keeps a fixed-size state; MLA keeps a latent for every token.



Kimi K3: KDA and MLA layers

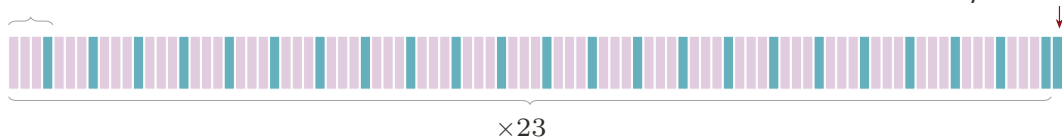
Kimi Team (2026)

K3 has 93 layers: 23 blocks of (KDA, KDA, KDA, MLA),
then a final MLA layer.

It has 2.8T parameters, of which 104B are active for each token.

3 KDA + 1 MLA

final MLA: Layer 93



69 KDA: Fixed-size states

24 MLA: Per-token latents

KDA keeps a fixed-size state; MLA keeps a latent for every token.

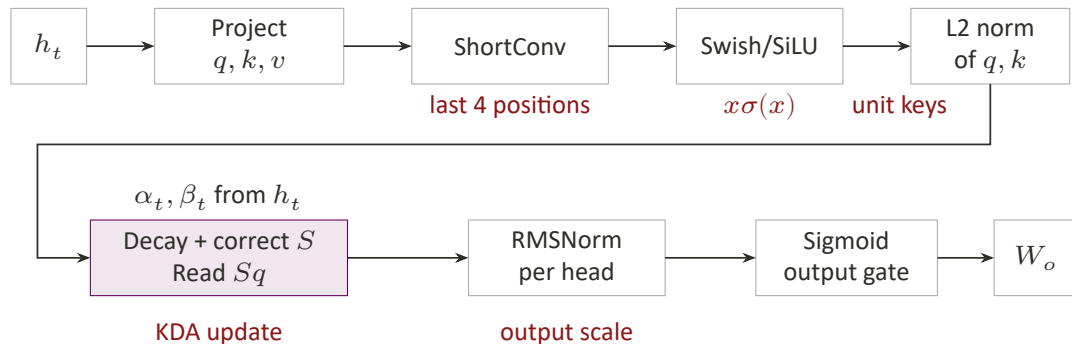
The MLA layers use no positional encoding.



Inside one Kimi K3 KDA layer

Source: Kimi Team (2026), Figure 2 and Equations (1)–(7).

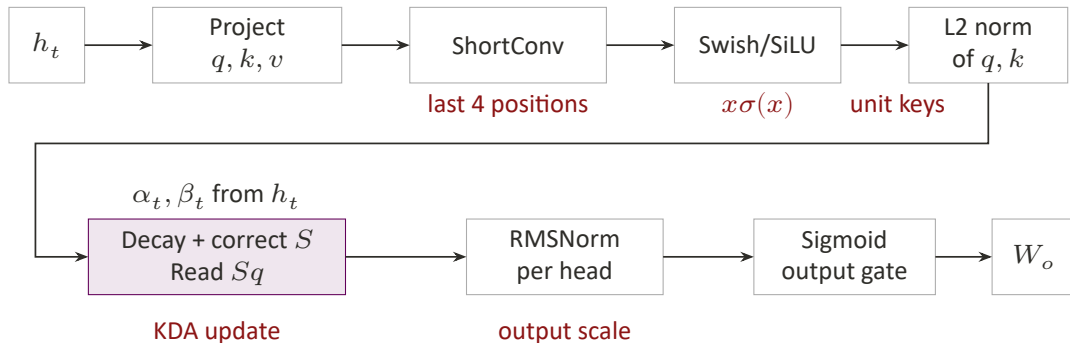
Combine these steps into a KDA layer.



Inside one Kimi K3 KDA layer

Source: Kimi Team (2026), Figure 2 and Equations (1)–(7).

Combine these steps into a KDA layer.



The weights of this read can be negative and need not sum to one. The normalization steps keep its scale under control instead.



Kimi K3: Memory as context grows

Count the numbers stored in all KDA states and MLA caches:

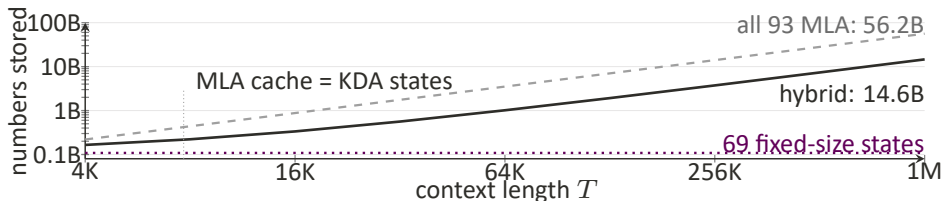
$$M(T) = 69s + 24cT$$



Kimi K3: Memory as context grows

Count the numbers stored in all KDA states and MLA caches:

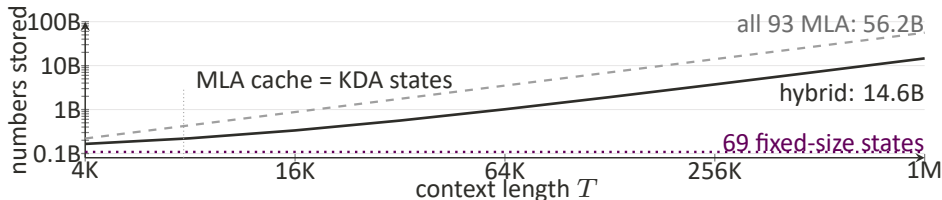
$$M(T) = 69s + 24cT$$



Kimi K3: Memory as context grows

Count the numbers stored in all KDA states and MLA caches:

$$M(T) = 69s + 24cT$$



Memory still grows with T , but only in few layers.



Recurrent layers in open models

Recent open models mix recurrent layers with full attention.

Gated DeltaNet

Qwen3-Next
(2025)

Qwen3.5
(2026)

KDA

Kimi Linear
(2025)

Kimi K3
(2026)

Mamba-2

Nemotron 3 Super
(2026)

Source: [Model cards and technical reports](#).

MS&E 319 Lecture 7



Summary

Positional information

Ensuring that embeddings carry positional information is important



Summary

Positional information

Ensuring that embeddings carry positional information is important
Several ways: Learned embeddings, RoPE, ALiBi, ...



Summary

Positional information

Ensuring that embeddings carry positional information is important
Several ways: Learned embeddings, RoPE, ALiBi, ...

Recurrent memory

Fixed-size recurrent state is an extreme form of KV-cache compression



Summary

Positional information

Ensuring that embeddings carry positional information is important
Several ways: Learned embeddings, RoPE, ALiBi, ...

Recurrent memory

Fixed-size recurrent state is an extreme form of KV-cache compression
Loses several useful properties of full softmax attention



Summary

Positional information

Ensuring that embeddings carry positional information is important
Several ways: Learned embeddings, RoPE, ALiBi, ...

Recurrent memory

Fixed-size recurrent state is an extreme form of KV-cache compression
Loses several useful properties of full softmax attention
Later methods (delta rule, decay, ...) recover some of these



Summary

Positional information

Ensuring that embeddings carry positional information is important
Several ways: Learned embeddings, RoPE, ALiBi, ...

Recurrent memory

Fixed-size recurrent state is an extreme form of KV-cache compression
Loses several useful properties of full softmax attention
Later methods (delta rule, decay, ...) recover some of these

Kimi K3 uses large fraction of recurrent layers; with few MLA layers



KDA: Decay + correct

Lecture 6 MLA; no RoPE



Questions?

