

Linear Attention

Presented by Malay, xyz
(as a Course Presentation for CS6013)

Primer: Kernel ML

Random Features for Large-Scale Kernel Machines
(Rahimi & Recht, 2007)

Transformers are RNNs
(Katharopoulos et al, 2020)

Rethinking Attention with Performers
(Choromanski et al., 2021)

Reformer: The Efficient Transformer
(Kitaev et al., 2020)

Linformer: Self-Attention with Linear Complexity
(Wang et al., 2020)

Primer: Kernel ML

Primer: positive-definite kernels

A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a **positive-definite (p.d.) kernel** if it is symmetric and every Gram matrix is PSD:

$$\sum_{i,j=1}^n c_i c_j k(x_i, x_j) \geq 0 \quad \forall n, x_1, \dots, x_n \in \mathcal{X}, c \in \mathbb{R}^n.$$

Examples

- Linear: $x^\top y$
- Polynomial: $(x^\top y + c)^p$
- Gaussian: $e^{-\|x-y\|^2/2\sigma^2}$
- Laplacian: $e^{-\|x-y\|_1/\sigma}$

Theorem (Moore - Aronszajn theorem)

k is positive-definite $\iff$ there exist a Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \rightarrow \mathcal{H}$ with

$$k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle_{\mathcal{H}}$$

Kernel trick: any algorithm that uses only inner products (SVM, ridge, PCA) can run in $\mathcal{H}$ without forming ϕ .

Theorem (Mercer)

A continuous positive definite kernel k on a compact space $\mathcal{X}$ can be written as

$$k(x, y) = \sum_{i=1}^{\infty} \lambda_i e_i(x) e_i(y),$$

where $\lambda_i \geq 0$ are eigenvalues and e_i are orthonormal eigenfunctions of the integral operator T_k , defined as $(T_k f)(x) = \int_{\mathcal{X}} k(x, y) f(y) d\mu(y)$. The series converges uniformly.

Explicit feature map: $\phi(x) = (\sqrt{\lambda_1} e_1(x), \sqrt{\lambda_2} e_2(x), \dots)$

Theorem (Bochner)

A continuous shift-invariant kernel $k(x, y) = \kappa(x - y)$ on $\mathbb{R}^d$ is p.d. iff $\kappa(\delta) = \int_{\mathbb{R}^d} e^{j\omega^\top \delta} d\Lambda(\omega)$ for a finite non-negative measure Λ , i.e., κ is the Fourier transform of Λ .

Explicit feature map: $\phi(x) = (e^{j\omega^\top x})_{\omega \in \mathbb{R}^d}$

Mercer vs. Bochner

	Mercer	Bochner
Domain	compact $\mathcal{X}$, measure μ	all of $\mathbb{R}^d$, shift-invariant k
Basis	eigenfunctions e_i	Fourier modes $e^{j\omega^\top x}$
Spectrum	discrete $\lambda_i \geq 0$	measure $\Lambda(\omega) \geq 0$
Feature map	$(\sqrt{\lambda_i} e_i(x))_{i \in \mathbb{N}}$	$(e^{j\omega^\top x})_{\omega \in \mathbb{R}^d}$, weighted by Λ

Random Features for Large-Scale Kernel Machines (Rahimi & Recht, 2007)

Why Kernel machines don't scale

Setup: N training points $x_1, \dots, x_N \in \mathbb{R}^d$ with labels $y_1, \dots, y_N$, and a shift-invariant kernel $k(x, y) = \kappa(x - y)$.

Kernel ridge regression / SVM learns $f(x) = \sum_{i=1}^N \alpha_i k(x_i, x)$, with Gram matrix $\mathbf{K}_{ij} = k(x_i, x_j)$.

- Gram matrix: $\mathcal{O}(N^2)$ memory & up to $\mathcal{O}(N^3)$ time for solving.
- Prediction: $\mathcal{O}(Nd)$ per test point (must keep all training points).

Idea

Find an explicit, low-dimensional, randomised map $z : \mathbb{R}^d \rightarrow \mathbb{R}^D$ with $k(x, y) \approx z(x)^\top z(y)$, then train a *linear* model on $z(x)$: $\mathcal{O}(ND)$ per training pass, $\mathcal{O}(Dd)$ prediction.

Random Fourier features: the kernel as an expectation

By Bochner's theorem, if κ is scaled so that $\kappa(0) = 1$, its Fourier transform $p(\omega)$ is a probability density. So, writing κ as IFT of p :

$$\kappa(x - y) = \int_{\mathbb{R}^d} p(\omega) e^{j\omega^\top (x-y)} d\omega = \mathbb{E}_{\omega \sim p}[\zeta_\omega(x) \zeta_\omega(y)^*], \quad \zeta_\omega(x) = e^{j\omega^\top x}$$

So $\zeta_\omega(x)\zeta_\omega(y)^*$ with $\omega \sim p$ is an unbiased estimate of $k(x, y)$.

Which distribution $p(\omega)$?

Kernel	$\kappa(\Delta)$	$p(\omega)$	
Gaussian	$e^{-\ \Delta\ _2^2/2}$	$(2\pi)^{-d/2}e^{-\ \omega\ _2^2/2}$	(Normal)
Laplacian	$e^{-\ \Delta\ _1}$	$\prod_i \frac{1}{\pi(1+\omega_i^2)}$	(Cauchy)
Cauchy	$\prod_i \frac{2}{1+\Delta_i^2}$	$e^{-\ \omega\ _1}$	(Laplace)

Making it real-valued

κ and p are real, so $e^{j\omega^\top(x-y)}$ can be replaced by $\cos \omega^\top(x-y)$:

$$z_\omega(x) = \begin{bmatrix} \cos(\omega^\top x) \\ \sin(\omega^\top x) \end{bmatrix} \Rightarrow z_\omega(x)^\top z_\omega(y) = \cos(\omega^\top(x-y)), \quad \mathbb{E}_\omega[z_\omega(x)^\top z_\omega(y)] = \kappa(x-y).$$

To reduce variance, concatenate D draws, scaled by $\sqrt{1/D}$:

$$z(x) = \sqrt{\frac{1}{D}} \begin{bmatrix} \cos(\omega_1^\top x) \\ \vdots \\ \cos(\omega_D^\top x) \\ \sin(\omega_1^\top x) \\ \vdots \\ \sin(\omega_D^\top x) \end{bmatrix}$$

$$z(x) \in \mathbb{R}^{2D},$$
$$z(x)^\top z(y) = \frac{1}{D} \sum_{j=1}^D \cos(\omega_j^\top(x-y)) \approx k(x,y)$$

How many features? Uniform convergence

Each term is bounded in $[-1, 1]$, so for a *fixed* pair Hoeffding gives $\Pr [|z(x)^\top z(y) - k(x, y)| \geq \epsilon] \leq 2 \exp(-D\epsilon^2/2)$.

Claim: Uniform Convergence of Fourier features

Let $\mathcal{M} \subset \mathbb{R}^d$ be compact with diameter $\text{diam}(\mathcal{M})$. Then

$$\Pr \left[\sup_{x, y \in \mathcal{M}} |z(x)^\top z(y) - k(x, y)| \geq \epsilon \right] \leq 2^8 \left(\frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon} \right)^2 \exp \left(-\frac{D\epsilon^2}{4(d+2)} \right)$$

where $\sigma_p^2 \equiv \mathbb{E}_{\omega \sim p}[\omega^\top \omega] = -\text{tr} \nabla^2 \kappa(0)$, the curvature of κ at the origin.

- So $D = \Omega\left(\frac{d}{\epsilon^2} \log \frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon}\right)$ gives error $\leq \epsilon$ *everywhere* on $\mathcal{M}$ w.h.p.
- Proof sketch: Hoeffding + union bound at the centres of an ϵ -net, & Markov's inequality on the Lipschitz constant of the error ($\mathbb{E}[L_f^2] \leq \sigma_p^2$) to extend between centres.

Algorithm: Random Fourier Features

Random Fourier Features

Input: positive-definite shift-invariant $k(x, y) = \kappa(x - y)$, number of features D

1. Compute $p(\omega)$, the Fourier transform of κ
2. Draw $\omega_1, \dots, \omega_D \stackrel{\text{iid}}{\sim} p$
3. Map every point: $z(x) = \sqrt{1/D} [\cos(\omega_j^\top x); \sin(\omega_j^\top x)]_{j=1}^D \in \mathbb{R}^{2D}$
4. Fit a *linear* method on $\{z(x_n)\}$
(The paper uses ridge regression $\min_w \|Z^\top w - y\|_2^2 + \lambda \|w\|_2^2$)

- Featurising N points: $\mathcal{O}(NDd)$. Ridge regression: $\mathcal{O}(ND^2 + D^3)$.
- Prediction: $w^\top z(x)$ in $\mathcal{O}(Dd)$, independent of N .

Transformers are RNNs

(Katharopoulos et al, 2020)

Recap: softmax attention

Input $x \in \mathbb{R}^{N \times d_{\text{model}}}$; projections $Q = xW_Q$, $K = xW_K \in \mathbb{R}^{N \times d_k}$, $V = xW_V \in \mathbb{R}^{N \times d_v}$.

$$O = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

Forming $QK^\top$ matrix takes $\mathcal{O}(N^2 \max(d_k, d_v))$ time & $\mathcal{O}(N^2)$ memory.

Attention as a generalised kernel smoother

Write row i of the output with an arbitrary similarity function:

$$O_i = \frac{\sum_{j=1}^N \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^N \text{sim}(Q_i, K_j)}$$

- Softmax attention: $\text{sim}(q, k) = \exp(q^\top k / \sqrt{d_k})$
- Only requirement for a valid attention: $\text{sim} \geq 0$

Substituting a feature map $\phi : \mathbb{R}^{d_k} \rightarrow \mathbb{R}^{d_\phi}$:

$$O_i = \frac{\sum_j \phi(Q_i)^\top \phi(K_j) V_j}{\sum_j \phi(Q_i)^\top \phi(K_j)} = \frac{\phi(Q_i)^\top \overbrace{\sum_j \phi(K_j) V_j^\top}^{d_\phi \times d_v}}{\underbrace{\phi(Q_i)^\top \sum_j \phi(K_j)}_{d_\phi}}$$

The $N \times N$ matrix is never materialised; cost is **linear in N** .

Choosing the feature map

- softmax kernel $\exp(q^\top k)$ has an **infinite-dimensional** feature map, so it cannot be linearised exactly.
- The paper's choice, applied elementwise:

$$\phi(x) = \text{elu}(x) + 1 = \begin{cases} x + 1 & x > 0 \\ e^x & x \leq 0 \end{cases}$$

- strictly positive, so the denominator never vanishes;
- non-zero gradients for negative inputs (unlike ReLU).

Note: ϕ is chosen for convenience, *not* to approximate softmax.

Causal masking

Autoregressive models: position i may only attend to $j \leq i$.

$$O_i = \frac{\phi(Q_i)^\top \sum_{j=1}^i \phi(K_j) V_j^\top}{\phi(Q_i)^\top \sum_{j=1}^i \phi(K_j)} = \frac{\phi(Q_i)^\top S_i}{\phi(Q_i)^\top Z_i}$$

with prefix sums

$$S_i = \sum_{j=1}^i \phi(K_j) V_j^\top \in \mathbb{R}^{d_\phi \times d_v}, \quad Z_i = \sum_{j=1}^i \phi(K_j) \in \mathbb{R}^{d_\phi}$$

- $S_i = S_{i-1} + \phi(K_i) V_i^\top$ and $Z_i = Z_{i-1} + \phi(K_i)$: each is an $\mathcal{O}(1)$ -in- N update.
- Whole sequence: $\mathcal{O}(N)$ instead of $\mathcal{O}(N^2)$.

Training memory: Gradients as Cumulative Sums

Storing every S_i for backprop would cost $\mathcal{O}(Nd_\phi d_v)$ memory.

Instead, for the numerator $\bar{O}_i = \phi(Q_i)^\top S_i$, with gradient $\nabla_{\bar{O}_i} \mathcal{L} \in \mathbb{R}^{1 \times d_v}$, define the *suffix* sum $R_i = \sum_{j=i}^N \phi(Q_j) \nabla_{\bar{O}_j} \mathcal{L} \in \mathbb{R}^{d_\phi \times d_v}$. Now,

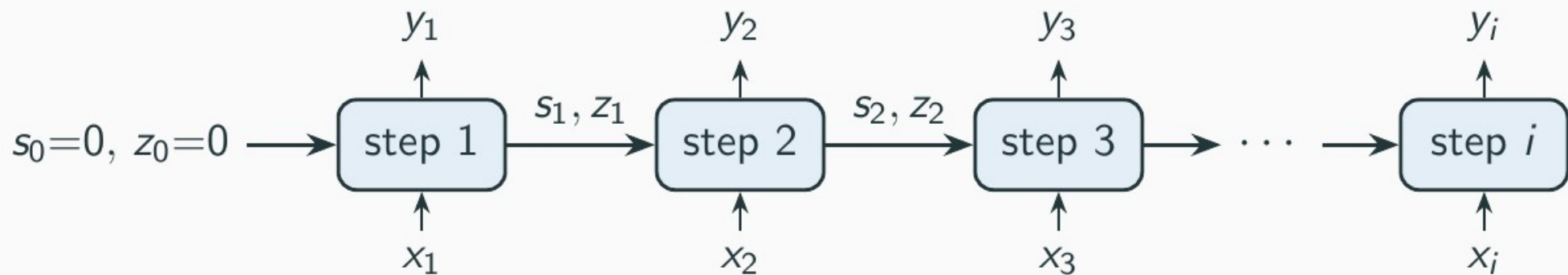
$$\nabla_{\phi(Q_i)} \mathcal{L} = \nabla_{\bar{O}_i} \mathcal{L} S_i^\top$$

$$\nabla_{\phi(K_i)} \mathcal{L} = R_i V_i$$

$$\nabla_{V_i} \mathcal{L} = R_i^\top \phi(K_i)$$

- Forward pass: running prefix sum S_i . Backward pass: running suffix sum R_i
- Time $\mathcal{O}(Nd_\phi d_v)$, memory $\mathcal{O}(N \max(d_\phi, d_v))$

... which is exactly an RNN



$$s_i = s_{i-1} + \phi(x_i W_K) (x_i W_V)^\top, \quad z_i = z_{i-1} + \phi(x_i W_K),$$

$$y_i = f_l \left(\frac{\phi(x_i W_Q)^\top s_i}{\phi(x_i W_Q)^\top z_i} + x_i \right)$$

f_l : the layer's position-wise feedforward network, as in a transformer.

- Hidden state (s_i, z_i) has **fixed size** $d_\phi \times d_v + d_\phi$: no growing KV cache.
- Train in parallel like a transformer; generate like an RNN with constant time and memory per token.

Complexity at a glance

	Training (whole sequence)		Generation (per token)	
	Time	Memory	Time	Memory
Softmax attention	$\mathcal{O}(N^2 \max(d_k, d_v))$	$\mathcal{O}(N^2 + N \max(d_k, d_v))$	$\mathcal{O}(N \max(d_k, d_v))$	$\mathcal{O}(N \max(d_k, d_v))$
Linear attention	$\mathcal{O}(Nd_\phi d_v)$	$\mathcal{O}(N \max(d_\phi, d_v))$	$\mathcal{O}(d_\phi d_v)$	$\mathcal{O}(d_\phi d_v)$

Experiments: image generation and speech

MNIST (784 pixels)			CIFAR-10 (3072 values)			Speech, WSJ (CTC loss)		
Method	Bits/dim	Images/s	Method	Bits/dim	Images/s	Method	PER (%)	s/epoch
Softmax	0.621	0.45 (1×)	Softmax	3.47	0.004 (1×)	Bi-LSTM	10.94	1047
LSH-1	0.745	0.68 (1.5×)	LSH-1	3.39	0.015 (3.75×)	Softmax	5.12	2711
LSH-4	0.676	0.27 (0.6×)	LSH-4	3.51	0.005 (1.25×)	LSH-4	9.33	2250
Linear	0.644	142.8 (317×)	Linear	3.40	17.85 (4,462×)	Linear	8.08	824

- Images: bits/dim close to softmax; generation is constant memory per pixel, since only (s_i, z_i) is stored.
- CIFAR-10: every model trained for 1 week on one GPU; the fast models fit more epochs, so beat softmax on bits/dim.
- Speech: softmax still has the best error, but linear beats Bi-LSTM and Reformer and is over 3× faster per epoch than softmax.

Linear Attention

Rethinking Attention with Performers

Presented by Parthiv

(as a Course Presentation for CS6013)

The attention output and the computational problem

For L query–key–value triples, scaled dot-product attention returns

$$o_i = \frac{\sum_{j=1}^L \exp(q_i^\top k_j / \sqrt{d}) v_j}{\sum_{j=1}^L \exp(q_i^\top k_j / \sqrt{d})}.$$

Set $x_i = d^{-1/4} q_i$, $y_j = d^{-1/4} k_j$, and $\text{SM}(x, y) = e^{x^\top y}$. The same softmax kernel appears in the numerator and denominator.

The conventional computation forms L^2 scores. Performer instead seeks $\text{SM}(x, y) \approx \phi(x)^\top \phi(y)$, so the sum over keys can be evaluated in a small feature space. We first derive the feature estimators, then substitute them back into the attention output.

Source: Secs. 2.1–2.3, Eqs. (1), (3), and (6).

Trigonometric features: the starting point

For a Gaussian projection, the characteristic-function identity gives

$$\mathbb{E}_{\omega \sim \mathcal{N}(0, I_d)} [\cos(\omega^\top u)] = e^{-\|u\|^2/2}.$$

Rewriting $x^\top y$ using $\|x - y\|^2$ turns softmax into a Gaussian kernel times separate norm corrections. The cosine subtraction formula then separates the two inputs.

The limitation is signed estimates.

Finite sine/cosine features can estimate a positive score by a negative number. Row normalization can then produce negative attention weights or a nearly cancelling denominator.

Source: Sec. 2.3, Eqs. (5)–(6).

Trigonometric estimator: complete derivation

Fix $x, y \in \mathbb{R}^d$. Let $C_{xy} = e^{(\|x\|^2 + \|y\|^2)/2}$ and $\omega \sim \mathcal{N}(0, I_d)$.

$$\begin{aligned} \text{SM}(x, y) &= e^{x^T y} \\ &= \exp\left(\frac{\|x\|^2 + \|y\|^2 - \|x - y\|^2}{2}\right) \\ &= C_{xy} \exp\left(-\frac{\|x - y\|^2}{2}\right) \\ &= C_{xy} \mathbb{E}_{\omega}[\cos(\omega^T (x - y))] \\ &= C_{xy} \mathbb{E}_{\omega}[\cos(\omega^T x) \cos(\omega^T y) + \sin(\omega^T x) \sin(\omega^T y)] \end{aligned}$$

Use $\cos(a - b) = \cos(a) \cos(b) + \sin(a) \sin(b)$ to separate the inputs.

Source: Sec. 2.3, Eqs. (5)–(6).

Trigonometric estimator: finite feature map

Draw $\omega_1, \dots, \omega_m \stackrel{\text{iid}}{\sim} \mathcal{N}(0, I_d)$. Use the same draw for both inputs.

$$\begin{aligned}\widehat{\text{SM}}_m^{\text{trig}}(x, y) &:= \frac{C_{xy}}{m} \sum_{a=1}^m \cos(\omega_a^T (x - y)) \\ &= \frac{C_{xy}}{m} \sum_{a=1}^m [\cos(\omega_a^T x) \cos(\omega_a^T y) \\ &\quad + \sin(\omega_a^T x) \sin(\omega_a^T y)]\end{aligned}$$

$$\phi_{\text{trig}}(x) := \frac{e^{\|x\|^2/2}}{\sqrt{m}} \begin{pmatrix} (\cos(\omega_a^T x))_{a=1}^m \\ (\sin(\omega_a^T x))_{a=1}^m \end{pmatrix}$$

$$\widehat{\text{SM}}_m^{\text{trig}}(x, y) = \phi_{\text{trig}}(x)^T \phi_{\text{trig}}(y), \quad \mathbb{E}[\widehat{\text{SM}}_m^{\text{trig}}(x, y)] = e^{x^T y}$$

The feature dimension is $r = 2m$. Averaging preserves the kernel expectation.

Source: Sec. 2.3, Eq. (5).

Positive features: a different identity

The alternative norm identity changes the route to the same target:

$$2x^T y = \|x + y\|^2 - \|x\|^2 - \|y\|^2.$$

For $\omega \sim \mathcal{N}(0, I_d)$, the real Gaussian moment identity is

$$\mathbb{E}[e^{\omega^T u}] = e^{\|u\|^2/2}.$$

The sum $x + y$ matches the positive exponent in this identity. Multiplying by $e^{-(\|x\|^2 + \|y\|^2)/2}$ cancels the individual squared norms. Splitting the remaining exponential produces one positive feature for each input.

Source: Lemma 1, Eq. (7). Gaussian moment identity used as a standard fact.

Positive estimator: complete derivation

Write $\Lambda = e^{-(\|x\|^2 + \|y\|^2)/2}$ and $\psi_\omega(t) = e^{\omega^\top t - \|t\|^2/2}$.

$$\begin{aligned} \text{SM}(x, y) &= e^{x^\top y} \\ &= \exp\left(\frac{\|x + y\|^2 - \|x\|^2 - \|y\|^2}{2}\right) \\ &= \Lambda \exp\left(\frac{\|x + y\|^2}{2}\right) \\ &= \Lambda \mathbb{E}_\omega[e^{\omega^\top (x+y)}] \\ &= \mathbb{E}_\omega\left[e^{\omega^\top x - \|x\|^2/2} e^{\omega^\top y - \|y\|^2/2}\right] \\ &= \mathbb{E}_\omega[\psi_\omega(x)\psi_\omega(y)], \quad \psi_\omega(x), \psi_\omega(y) > 0 \end{aligned}$$

The separated integrand is a product of strictly positive functions.

Source: Lemma 1, Eq. (7).

Positive estimator: finite feature map

For m iid Gaussian projections, retain $\psi_\omega(t) = e^{\omega^\top t - \|t\|^2/2}$.

$$\widehat{\text{SM}}_m^+(x, y) := \frac{1}{m} \sum_{a=1}^m \psi_{\omega_a}(x) \psi_{\omega_a}(y)$$

$$\phi_+(x) := \frac{1}{\sqrt{m}} (\psi_{\omega_1}(x), \dots, \psi_{\omega_m}(x))^\top \in \mathbb{R}^m$$

$$\widehat{\text{SM}}_m^+(x, y) = \phi_+(x)^\top \phi_+(y) > 0$$

$$\mathbb{E}[\widehat{\text{SM}}_m^+(x, y)] = \frac{1}{m} \sum_{a=1}^m \mathbb{E}[\psi_{\omega_a}(x) \psi_{\omega_a}(y)] = \text{SM}(x, y)$$

Each summand has the correct mean, so the kernel estimate remains unbiased.

Positive scores give a positive row sum and convex attention weights in exact arithmetic. The normalized output, being a ratio, need not be unbiased.

Source: Lemma 1 and Eq. (5).

Hyperbolic features: pairing the two signs

Write $z = x + y$ and $\Lambda = e^{-(\|x\|^2 + \|y\|^2)/2}$. An ordinary positive sample contributes $\Lambda e^{\omega^T z}$. Gaussian symmetry makes ω and $-\omega$ identically distributed. Average their two estimates using the *same* sampled direction:

$$\frac{e^{\omega^T z} + e^{-\omega^T z}}{2} = \cosh(\omega^T z).$$

The estimator stays positive and unbiased. The dependence within each pair reduces variance. There are m sampled directions and $2m$ feature coordinates.

Source: Lemma 1, Eq. (7), and Lemma 2.

Hyperbolic estimator: complete derivation

Let $\Omega = (\omega_1, \dots, \omega_m)$, $z = x + y$, and $\Lambda = e^{-(\|x\|^2 + \|y\|^2)/2}$.

$$\begin{aligned}\widehat{\text{SM}}_m^+(\Omega; x, y) &= \frac{\Lambda}{m} \sum_{a=1}^m e^{\omega_a^T z} \\ \widehat{\text{SM}}_m^{\text{hyp}+}(x, y) &:= \frac{1}{2} \left[\widehat{\text{SM}}_m^+(\Omega; x, y) + \widehat{\text{SM}}_m^+(-\Omega; x, y) \right] \\ &= \frac{\Lambda}{2m} \sum_{a=1}^m \left(e^{\omega_a^T z} + e^{-\omega_a^T z} \right) \\ &= \frac{\Lambda}{m} \sum_{a=1}^m \cosh(\omega_a^T z) \\ &= \frac{1}{2m} \sum_{a=1}^m \left[\psi_{\omega_a}(x) \psi_{\omega_a}(y) + \psi_{-\omega_a}(x) \psi_{-\omega_a}(y) \right]\end{aligned}$$

Split each sign term into a positive feature product for x and y .

Source: Lemma 1, Eq. (7), written as antithetic pairing of the positive estimator.

Hyperbolic estimator: feature map and mean

The base directions $\omega_1, \dots, \omega_m$ are iid. The signs within each pair share a direction.

$$\phi_{\text{hyp}+}(x) := \frac{1}{\sqrt{2m}} \begin{pmatrix} (\psi_{\omega_a}(x))_{a=1}^m \\ (\psi_{-\omega_a}(x))_{a=1}^m \end{pmatrix}$$

$$\begin{aligned} \phi_{\text{hyp}+}(x)^\top \phi_{\text{hyp}+}(y) &= \frac{1}{2m} \sum_{a=1}^m [\psi_{\omega_a}(x)\psi_{\omega_a}(y) + \psi_{-\omega_a}(x)\psi_{-\omega_a}(y)] \\ &= \widehat{\text{SM}}_m^{\text{hyp}+}(x, y) > 0 \end{aligned}$$

$$\mathbb{E}[\widehat{\text{SM}}_m^{\text{hyp}+}(x, y)] = \text{SM}(x, y), \quad r = 2m$$

The mean is unchanged. There are m sampled directions and $2m$ coordinates.

Source: Lemma 1 and Eq. (5).

Orthogonal features: spreading the directions

Independent projection directions can lie close to one another, giving partly redundant observations of the same input pair. Orthogonal sampling spreads the directions by requiring them to be perpendicular.

We keep the positive feature formula and change the joint sampling scheme. Each individual projection must retain distribution $\mathcal{N}(0, I_d)$ so that the estimator's expectation stays correct.

A standard Gaussian vector has a uniform direction and an independent χ_d length. Sample orthogonal uniform directions, then give each an independent Gaussian radius. This preserves the marginal Gaussian law while introducing dependence across projections.

Source: Sec. 2.4 and Appendix B.2.

Orthogonal estimator: construction and mean

Use $z = x + y$ and $\Lambda = e^{-(\|x\|^2 + \|y\|^2)/2}$. For one block, assume $m \leq d$.

$$\widehat{\text{SM}}_m^+(x, y) = \frac{\Lambda}{m} \sum_{a=1}^m e^{(\omega_a^{\text{iid}})^T z}, \quad \omega_a^{\text{iid}} \stackrel{\text{iid}}{\sim} \mathcal{N}(0, I_d)$$

$$\omega_a^{\text{ort}} := \rho_a u_a, \quad u_a^T u_b = 0, \quad \rho_a \stackrel{\text{iid}}{\sim} \chi_d$$

$$\widehat{\text{SM}}_m^{\text{ort}+}(x, y) := \frac{\Lambda}{m} \sum_{a=1}^m e^{(\omega_a^{\text{ort}})^T z}$$

$$\mathbb{E}[\widehat{\text{SM}}_m^{\text{ort}+}(x, y)] = \frac{\Lambda}{m} \sum_{a=1}^m \mathbb{E}[e^{(\omega_a^{\text{ort}})^T z}] = \text{SM}(x, y)$$

Each projection is marginally Gaussian. Linearity of expectation needs no independence.

Source: Sec. 2.4 and Appendix B.2. The frame and all radii are independent.

Linear attention: complete derivation

Fix the sampled map $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^r$. Use row outputs, $S \in \mathbb{R}^{r \times d_v}$, and $z \in \mathbb{R}^r$.

$$\begin{aligned} o_i^\top &= \frac{\sum_j \text{SM}(x_i, y_j) v_j^\top}{\sum_j \text{SM}(x_i, y_j)} \\ \hat{o}_i^\top &:= \frac{\sum_j \phi(x_i)^\top \phi(y_j) v_j^\top}{\sum_j \phi(x_i)^\top \phi(y_j)} \\ &= \frac{\phi(x_i)^\top \left(\sum_j \phi(y_j) v_j^\top \right)}{\phi(x_i)^\top \left(\sum_j \phi(y_j) \right)} \\ &= \frac{\phi(x_i)^\top S}{\phi(x_i)^\top z}, \quad S := \sum_j \phi(y_j) v_j^\top, \quad z := \sum_j \phi(y_j) \\ \hat{O} &= \text{diag}(\Phi_Q z)^{-1} \Phi_Q S, \quad S = \Phi_K^\top V, \quad z = \Phi_K^\top \mathbf{1}_L \end{aligned}$$

Stack all query outputs. Reassociation creates no additional approximation.

Source: Eq. (4), expanded into a compact derivation.

FAVOR+: computation and complexity

FAVOR+ combines positive orthogonal features with the reassocated attention computation. Its name is Fast Attention Via positive Orthogonal Random features.

$$S = \Phi_K^T V, \quad z = \Phi_K^T \mathbf{1}_L, \quad \hat{O} = \text{diag}(\Phi_Q z)^{-1} \Phi_Q S.$$

	Conventional exact attention	Feature attention
Time, $d_v = d$	$O(L^2 d)$	$O(Lrd)$
Space	$O(L^2 + Ld)$	$O(Lr + Ld + rd)$

For fixed r, d , cost grows linearly with L . The attention matrix is not materialized. A stricter accuracy requirement may require a larger feature budget. Orthogonal sampling adds an amortizable setup cost.

Source: Secs. 2.2–2.4, Algorithm 1, and Appendix B.3. Exact baseline materializes its score matrix.

Causal attention: complete prefix derivation

At position i , only $j \leq i$ is visible. Initialize $S_0 = 0$ and $z_0 = 0$.

$$\begin{aligned}\hat{o}_i^\top &= \frac{\sum_{j \leq i} \phi(x_i)^\top \phi(y_j) v_j^\top}{\sum_{j \leq i} \phi(x_i)^\top \phi(y_j)} \\ &= \frac{\phi(x_i)^\top S_i}{\phi(x_i)^\top z_i}, \quad S_i := \sum_{j \leq i} \phi(y_j) v_j^\top, \quad z_i := \sum_{j \leq i} \phi(y_j)\end{aligned}$$

$$S_i = S_{i-1} + \phi(y_i) v_i^\top$$

$$z_i = z_{i-1} + \phi(y_i)$$

Update the denominator summary with one vector.

Source: Appendix B.1, Eq. (11). Training memory also depends on the scan and gradient implementation.

The four softmax estimator choices

Estimator	Kernel target	Finite scores	Coordinates
Trigonometric	$e^{x^T y}$	May be signed	$2m$
Positive	$e^{x^T y}$	Positive	m
Hyperbolic positive	$e^{x^T y}$	Positive	$2m$
Orthogonal positive	$e^{x^T y}$	Positive	m

All estimate the same kernel. The constructions address different issues: separating the inputs, preserving positivity, pairing signs to reduce variance, and coupling directions to reduce variance further.

The following results state precisely what these choices guarantee. Kernel error, normalized-output error, and task quality remain distinct quantities.

Source: Sec. 2 and the theoretical results in Sec. 3.

Lemma 2: kernel error depends on geometry

Fix x, y , and define $K = e^{x^\top y}$, $s = \|x + y\|^2$, $t = \|x - y\|^2$. The estimators are unbiased, so $\text{MSE}(\hat{K}) = \mathbb{E}[(\hat{K} - K)^2] = \text{Var}(\hat{K})$.

$$\text{MSE}(\widehat{\text{SM}}_m^{\text{trig}}) = \frac{e^s K^{-2}}{2m} (1 - e^{-t})^2, \quad \text{MSE}(\widehat{\text{SM}}_m^+) = \frac{K^2}{m} (e^s - 1).$$

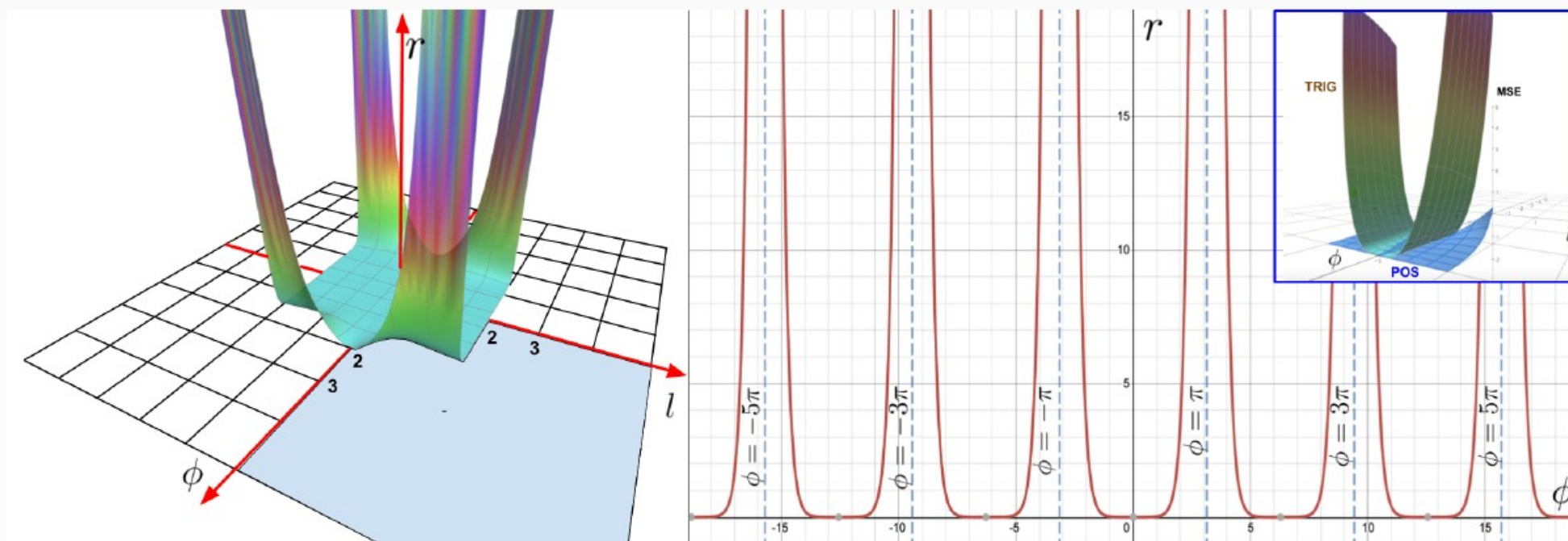
Opposing inputs. For $y = -x$, $s = 0$: positive features are exact, while trigonometric error can grow as $K \rightarrow 0$.

Aligned inputs. For $y = x$, $t = 0$: trigonometric features are exact, while positive features generally retain variance.

Both MSEs scale as $1/m$ for fixed inputs. These are *kernel-entry* guarantees, before normalization or propagation through a network.

Source: Lemma 2, Eq. (8).

Figure 2: where positive features help



The plotted ratio is $r_{\text{err}} = \text{MSE}(\text{trig}) / \text{MSE}(\text{positive})$. Values above one favor positive features. The left panel varies angle and norm; the right fixes the norm at one. At large angles, scores become small and positive features can be much more accurate. This is a geometry-dependent advantage, rather than a universal MSE ranking.

Source: Fig. 2 and Lemma 2. The plot's r denotes the error ratio.

Lemma 2: the gain from sign pairing

For $s = \|x + y\|^2$ and m independent direction pairs,

$$\text{MSE}(\widehat{\text{SM}}_m^{\text{hyp}+}) = \frac{1}{2}(1 - e^{-s}) \text{MSE}(\widehat{\text{SM}}_m^+).$$

The multiplier is in $[0, 1/2)$ for finite s . Pairing therefore reduces variance at the same number of sampled directions.

At the same feature dimension, compare with $2m$ iid samples:

$$\text{MSE}(\widehat{\text{SM}}_m^{\text{hyp}+}) = (1 - e^{-s}) \text{MSE}(\widehat{\text{SM}}_{2m}^+) \leq \text{MSE}(\widehat{\text{SM}}_{2m}^+).$$

The improvement is strict for $s > 0$. At $s = 0$, both errors are zero. The $2m$ coordinates reuse only m projected dot products.

Source: Lemma 2, Eq. (8), and the feature construction in Eq. (5).

Theorem 2: orthogonality reduces variance

Compare positive estimators at the same fixed x, y and sample count m . One orthogonal block requires $m \leq d$; every projection remains marginally $\mathcal{N}(0, I_d)$.

$$\text{MSE}(\widehat{\text{SM}}_m^{\text{ort}+}) \leq \text{MSE}(\widehat{\text{SM}}_m^+) - \frac{2(m-1)}{m(d+2)}(K - \Lambda)^2,$$

where $K = e^{x^\top y}$ and $\Lambda = e^{-(\|x\|^2 + \|y\|^2)/2}$. The guaranteed reduction is positive for $m > 1$ and $x + y \neq 0$. This is a finite-dimensional result at the same feature count. For $m > d$, use multiple orthogonal blocks.

Source: Theorem 2, Eq. (10), Sec. 2.4, and Appendix B.2.

Theorem 4: a feature budget independent of length

Assume $\|q_i\|, \|k_j\| \leq R$. Scaled inputs have norm at most $\ell = Rd^{-1/4}$. Set $h_* = e^{\ell^2/2}$ and $\delta = \varepsilon/h_*^2$.

$$m = \Theta\left(\frac{d}{\delta^2} \log \frac{4d^{3/4}R}{\delta}\right) \implies \max_{i,j} |\hat{A}_{ij} - A_{ij}| \leq \varepsilon$$

with a fixed constant success probability in the theorem's parameter regime.

Scope. The result concerns *trigonometric features with orthogonal projections* and the unnormalized attention matrix A . One map approximates all pairs over the bounded domain.

Implication. The budget depends on dimension, norm bound, and precision, with no explicit sequence-length factor L . It does not by itself guarantee normalized outputs or task accuracy.

Source: Theorem 4, Appendices F.5–F.6. Statement without proof.

Generalized attention: choosing another kernel

The fast computation only needs a separable nonnegative feature map. It is not tied to exponentials.

$$\mathcal{K}_{\text{ReLU}}(x, y) = \mathbb{E}_{\omega}[\text{ReLU}(\omega^{\top} x) \text{ReLU}(\omega^{\top} y)].$$

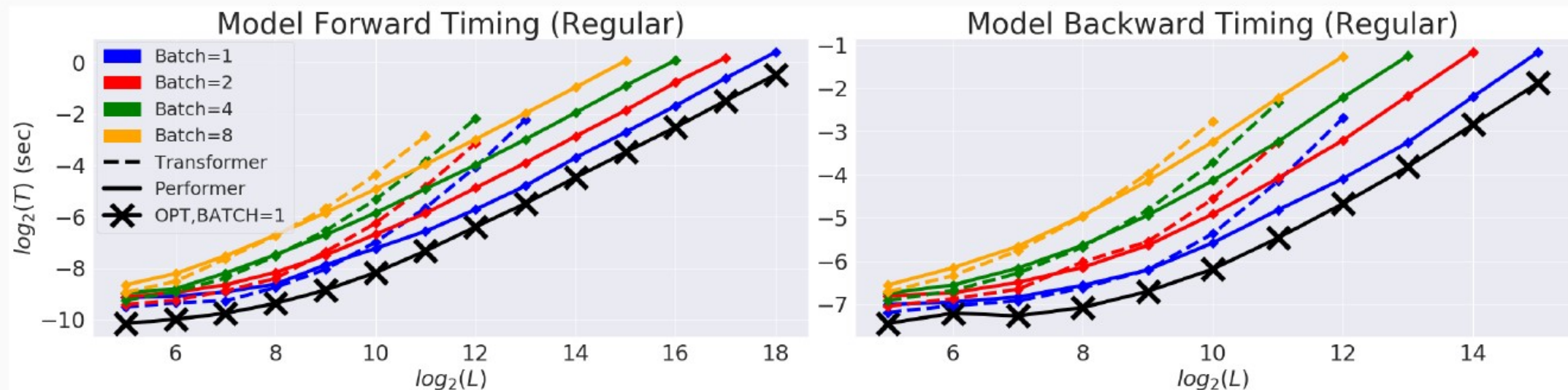
ReLU features define an alternative kernel. They do not estimate $e^{x^{\top} y}$. This distinction separates the successful softmax approximation from the broader Performer modeling framework.

The protein experiments show why this matters. Once several kernels become tractable at long context lengths, their task performance can be compared directly rather than assuming softmax is always best.

Source: Sec. 2.3, Eq. (5), Sec. 4.4, and Appendix D.3.

Experiments 1/3: computation at longer contexts

JAX/JIT, 16 GB V100; $(n_{\text{heads}}, n_{\text{layers}}, d_{\text{ff}}, d_{\text{model}}) = (8, 6, 2048, 512)$.

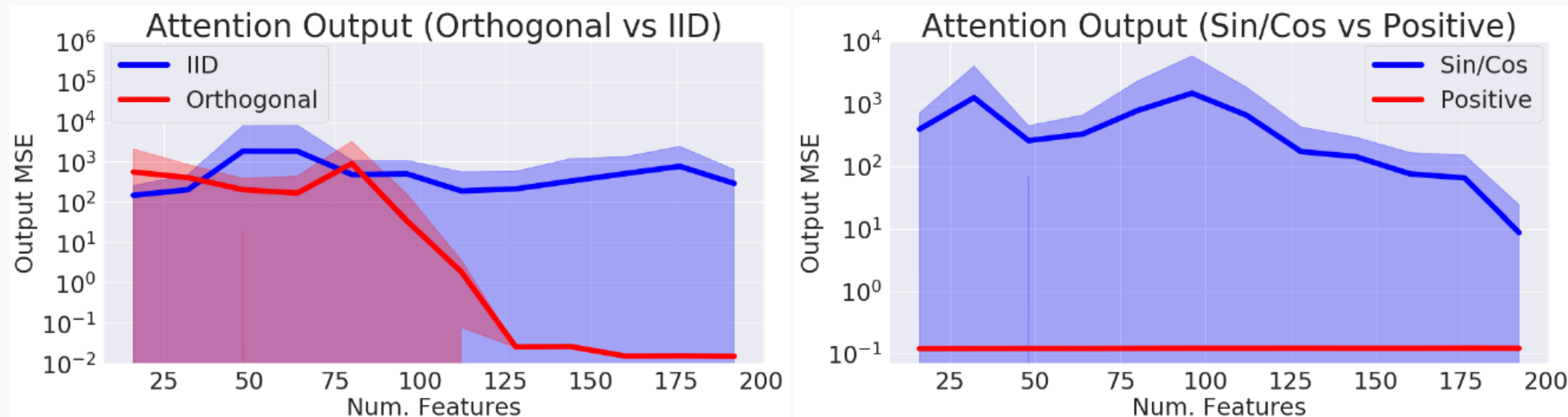


Performer scales more slowly with length and fits longer sequences than the materialized Transformer baseline. Left: forward time. Right: backward time. Both axes are logarithmic. OPT returns V directly and measures an idealized speed limit. The observed gains apply to the paper's implementations and hardware.

Source: Sec. 4.1 and Fig. 3.

Experiments 2/3: normalized attention-output error

Synthetic inputs: $L = 4096$, $d = 16$. Vary the random-feature sample budget.

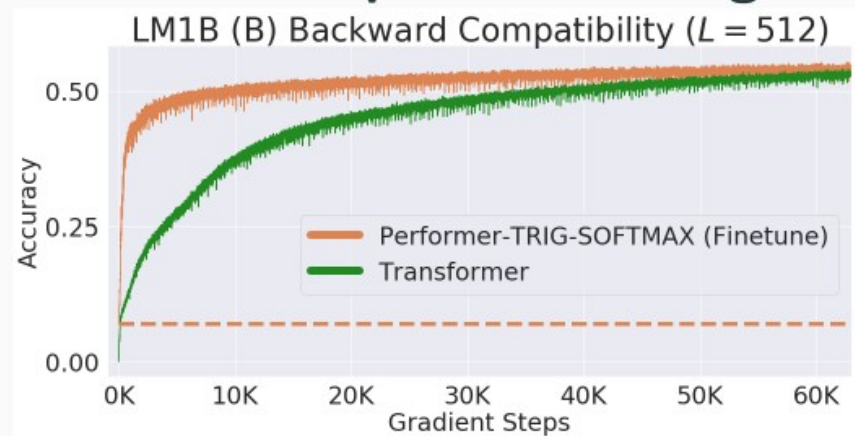


Orthogonal sampling gives lower output MSE than iid sampling (left). Positive features outperform sine/cosine features in the tested geometry (right). The metric includes normalization and values. Shading shows standard deviations over 15 random input matrices, rather than confidence intervals across training runs.

Source: Sec. 4.2 and Fig. 4.

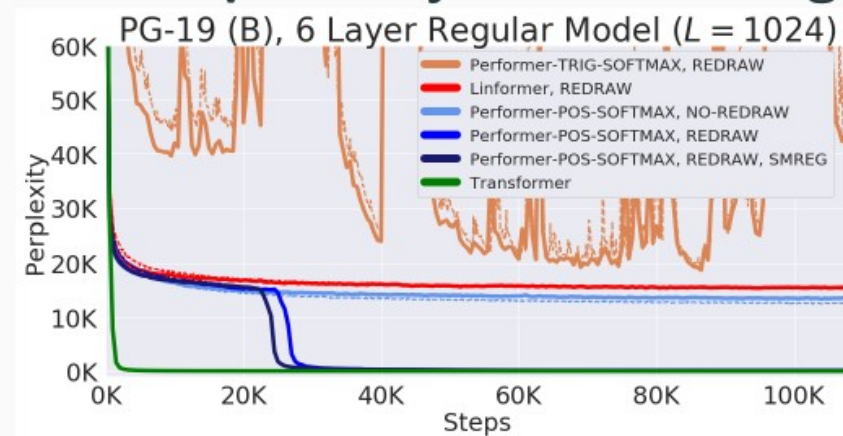
Experiments 3/3: transfer and stable text training

LM1B: reuse pretrained weights



Trigonometric features, $L = 512$: accuracy rises from about 0.07 and recovers with fine-tuning. Errors can grow through depth (Fig. 14).

PG-19: positivity and redrawing



At $L = 1024$, trigonometric training is unstable. Positive features with redrawing match Transformer perplexity; fixed features plateau higher.

Redrawing resamples the feature map during training, shared by queries and keys.

Source: Sec. 4.3, Fig. 5, and Appendix D.1, Fig. 14. Both text tasks use bidirectional prediction.

What to retain from Performer

The mathematical construction

Positive feature products estimate the original softmax kernel. Sign pairing and orthogonal directions reduce kernel variance. Reassociation converts pairwise attention into shared summaries.

The computational consequence

For fixed feature count and head dimension, attention cost grows linearly with sequence length. Prefix summaries also support causal inference.

The practical choices

Feature count trades cost for fidelity. Exponentials still require numerical care, and errors can propagate across layers. Fine-tuning, redrawing, and the kernel choice affect task quality.

Source: Choromanski et al., Rethinking Attention with Performers, ICLR 2021. Supplied version: arXiv:2009.14794v4.

Reformer

The Efficient Transformer

ICLR 2020

Presented by 25m0801 - Arghya

Course presentation for CS6013

Attention grows quadratically with sequence length

A sequence of length L creates one score matrix per attention head. Queries and keys have width d_k .

$$\underbrace{QK^T}_{\text{scores for one head}} \in \mathbb{R}^{L \times L}$$

For batch size β and n_h heads the main attention costs are

$$\text{Score storage} = O(\beta n_h L^2)$$

$$\text{Score computation} = O(\beta n_h L^2 d_k)$$

Doubling the sequence length gives four times as many scores. One float32 score matrix at $L = 65536$ occupies 16 GiB.

LSH attention reduces the candidate set for each query.

Stored activations grow with the number of layers

Backpropagation usually keeps earlier layer states until they are needed. With model width d_{model} each state has shape

$$X^{(\ell)} \in \mathbb{R}^{\beta \times L \times d_{\text{model}}}$$

Retaining these states across N layers gives

$$M_{\text{saved}} = O(N\beta L d_{\text{model}})$$

$$N \longrightarrow 2N \quad \implies \quad \text{twice the stored layer states}$$

This cost grows with depth even when a single layer fits in memory.

Reversible blocks recover earlier states during backpropagation.

Feed-forward layers create wide intermediate activations

Each position expands into a wider hidden vector before returning to model width.

$$\underbrace{\beta \times L \times d_{\text{model}}}_{\text{input}} \longrightarrow \underbrace{\beta \times L \times d_{\text{ff}}}_{\text{wide intermediate}} \longrightarrow \underbrace{\beta \times L \times d_{\text{model}}}_{\text{output}}$$

The wide intermediate alone needs

$$M_{\text{FFN}} = O(\beta L d_{\text{ff}}) \quad \text{per layer}$$

$$d_{\text{ff}} \approx 4d_{\text{model}} \implies \text{about four times the input storage}$$

Processing every position at once makes this a large temporary allocation.

Chunking processes fewer positions at once to lower peak workspace.

Dense attention mixes values using similarity

$$S = \frac{QK^{\top}}{\sqrt{d_k}} \quad O = \text{softmax}(S)V$$

- A query describes what a position is looking for.
- A key describes what a position can offer.
- Softmax turns each row of scores into weights on the values.

Dimensions

The sequence has length L . Each query and key has width d_k . The score matrix has shape $L \times L$ for each head.

What if queries and keys used the same projection?

Usually the activations A pass through two learned projections.

$$Q = AW_Q \quad K = AW_K$$

Instead we could share the projection.

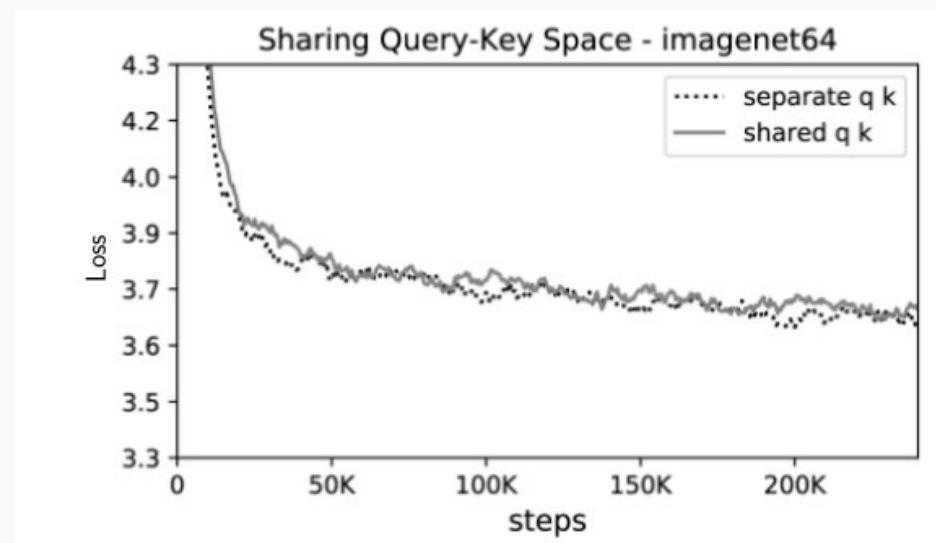
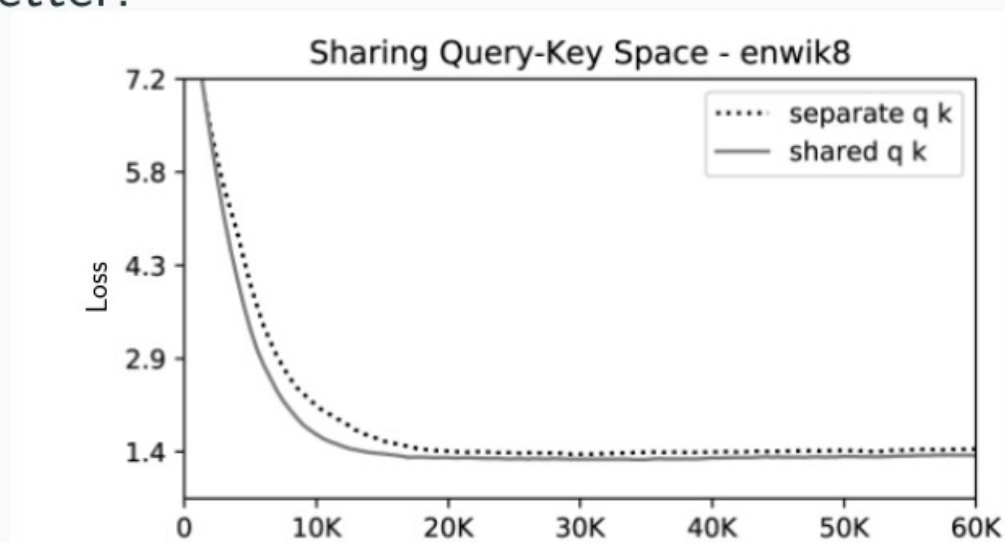
$$\underbrace{Q = K = AW_{QK}}_{\text{before key normalization}}$$

Queries and keys would share a single set of vectors. LSH for angular similarity could group nearby directions in that set.

Does sharing the projection change prediction quality?

Shared projections retain quality in these experiments

Three-layer models with eight heads. Loss is measured in bits per dimension. Lower is better.



- Shared and separate projections give comparable loss curves.

Shared directions turn attention into an angular search

Normalize each key while keeping the query magnitude.

$$q_i = a_i W_{QK} \quad k_j = \frac{q_j}{\|q_j\|} \quad v_j = a_j W_V$$

$$q_i \cdot k_j = \|q_i\| \cos \theta_{ij}$$

- The query norm stays fixed while we compare its target keys.
- The largest scores therefore come from the smallest angles.

We can use LSH for angular similarity to find likely attention targets.

Reformer uses LSH for angular similarity

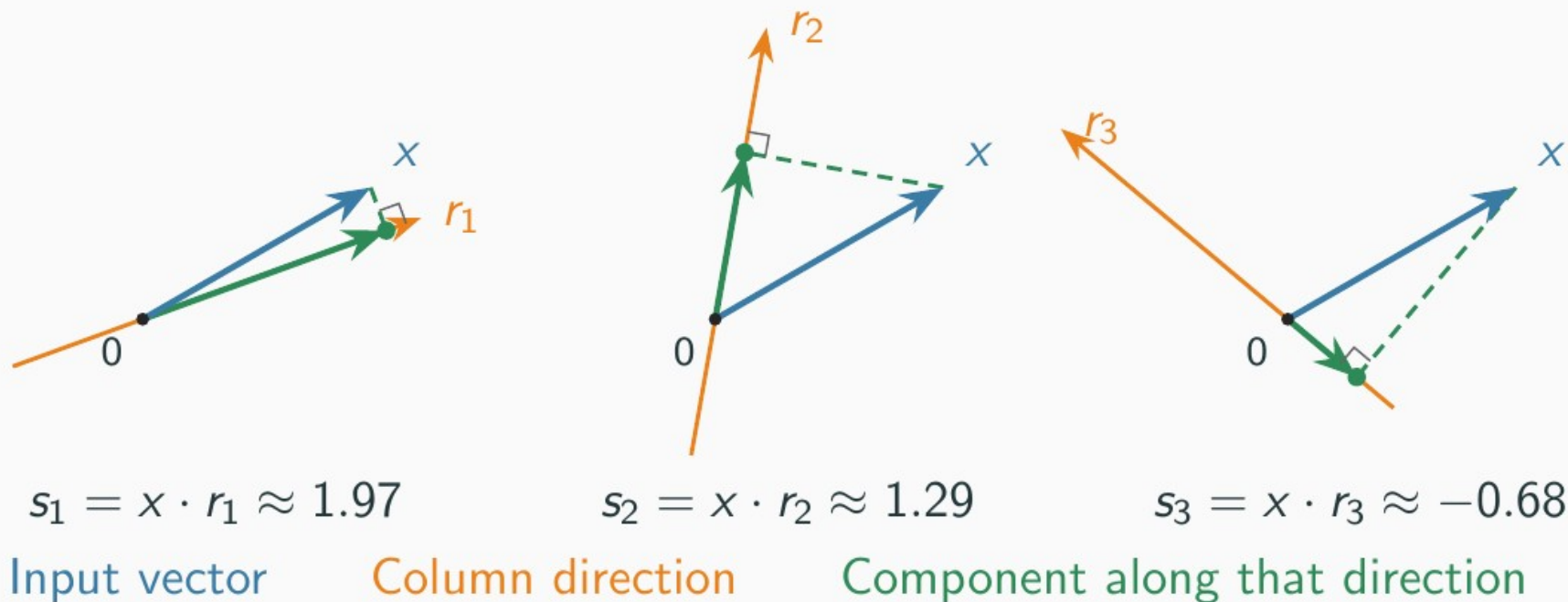
Let B be the number of possible bucket labels. Draw R_{proj} with independent standard Gaussian entries.

$$R_{\text{proj}} \in \mathbb{R}^{d_k \times B/2} \quad h(x) = \arg \max_{1 \leq i \leq B} \left(\begin{bmatrix} xR_{\text{proj}} & -xR_{\text{proj}} \end{bmatrix} \right)_i$$

1. Project each vector onto the random axes.
2. Include both signs of every projection.
3. Return the winning index i as the bucket label.

What does multiplying by a random matrix do?

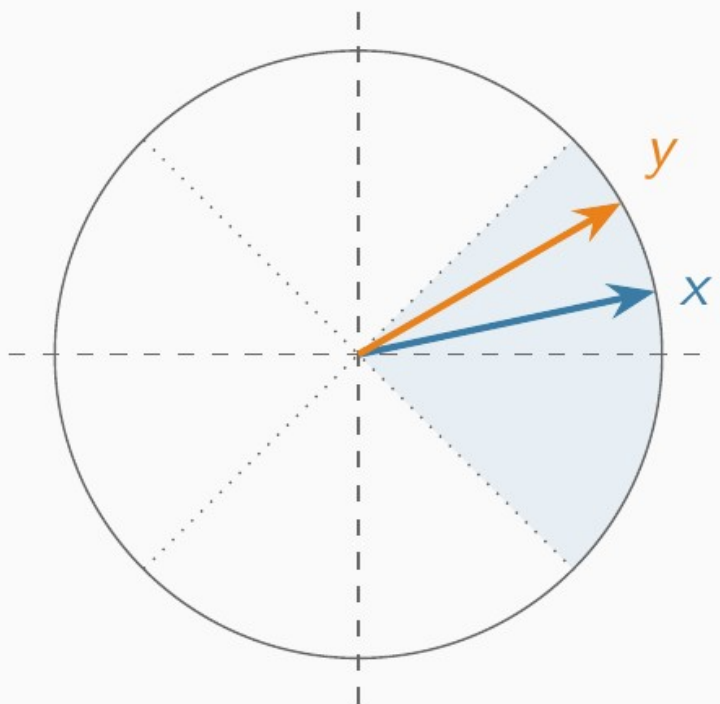
The columns of $R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$ give three directions. Each column produces one signed scalar response.



$$xR = \begin{bmatrix} x \cdot r_1 & x \cdot r_2 & x \cdot r_3 \end{bmatrix} \approx \begin{bmatrix} 1.97 & 1.29 & -0.68 \end{bmatrix}$$

Unit columns are shown. In general $x \cdot r_j = \|x\| \|r_j\| \cos \theta_j$.

Nearby directions often share a hash bucket



One possible random orientation

- A random projection defines angular regions.
- Each region receives a bucket label.
- Nearby vectors often land in the same region.
- A different projection can separate the same pair.

Standard LSH for angular similarity uses sign bits

Draw k independent Gaussian vectors $g_1 \dots g_k$ with standard Gaussian entries. Each direction is perpendicular to a hyperplane through the origin. Each projection contributes one positive or negative sign.

$$k = \log_2 B \quad h_{\text{bit}}(x) = \begin{bmatrix} \text{sign}(g_1^\top x) & \dots & \text{sign}(g_k^\top x) \end{bmatrix}$$

- A label with k bits has 2^k possible values.
- For $B = 1024$ one bucket label needs 10 binary projections.
- A new hash round draws a fresh set of those projections.

For example the four signs $+ \quad - \quad - \quad +$ can be stored as 1001.

Collision exponents compare the two hash rules

For a fixed angle $0 < \theta < \pi$ let $p(\theta)$ be the chance of a full label collision in one round.

$$p_{\text{bit}}(\theta) = B^{-\alpha_{\text{bit}}(\theta)} \quad \alpha_{\text{bit}}(\theta) = -\log_2 \left(1 - \frac{\theta}{\pi} \right)$$

$$p_{\text{arg}}(\theta) = B^{-\alpha_{\text{arg}}(\theta) + o(1)} \quad \alpha_{\text{arg}}(\theta) = \tan^2(\theta/2)$$

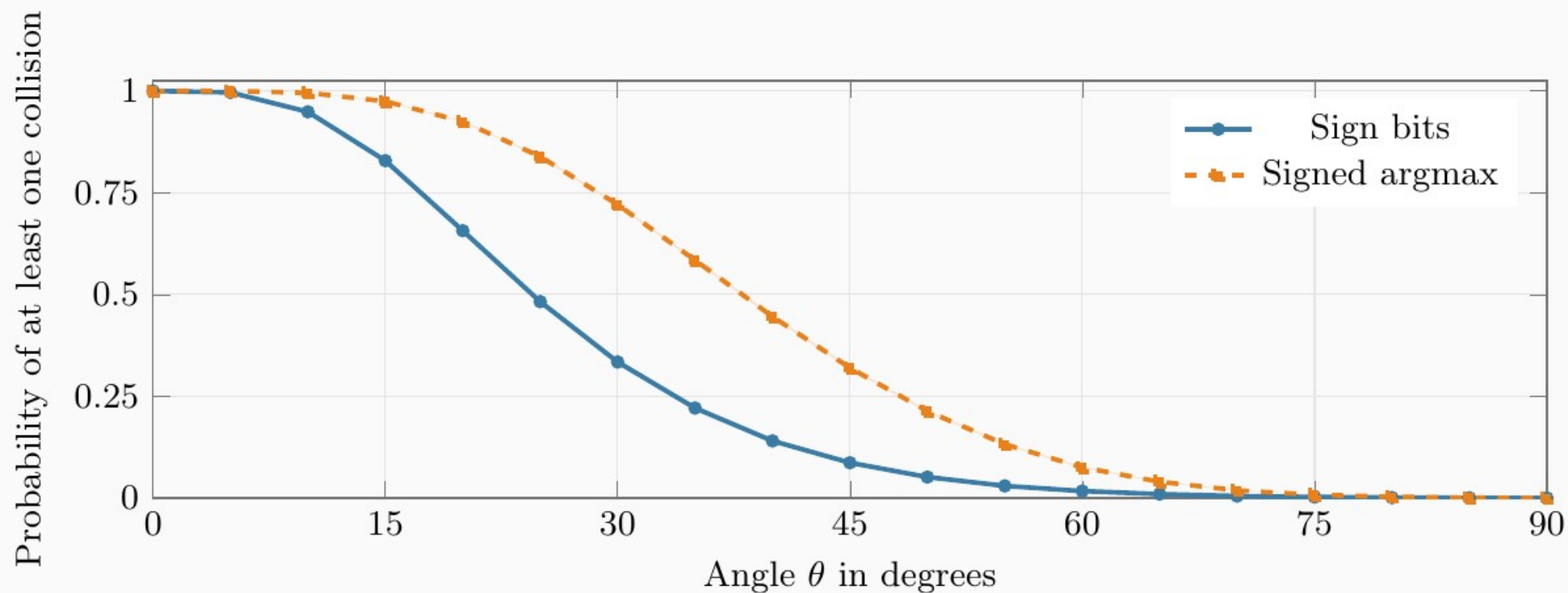
A smaller alpha means slower collision decay as B grows.

$$\alpha_{\text{bit}}(\theta) = \frac{\theta}{\pi \ln 2} + O(\theta^2) \quad \alpha_{\text{arg}}(\theta) = \frac{\theta^2}{4} + O(\theta^4)$$

Near zero the leading growth is linear for sign bits and quadratic for signed argmax.

Collision probability across five hashing rounds

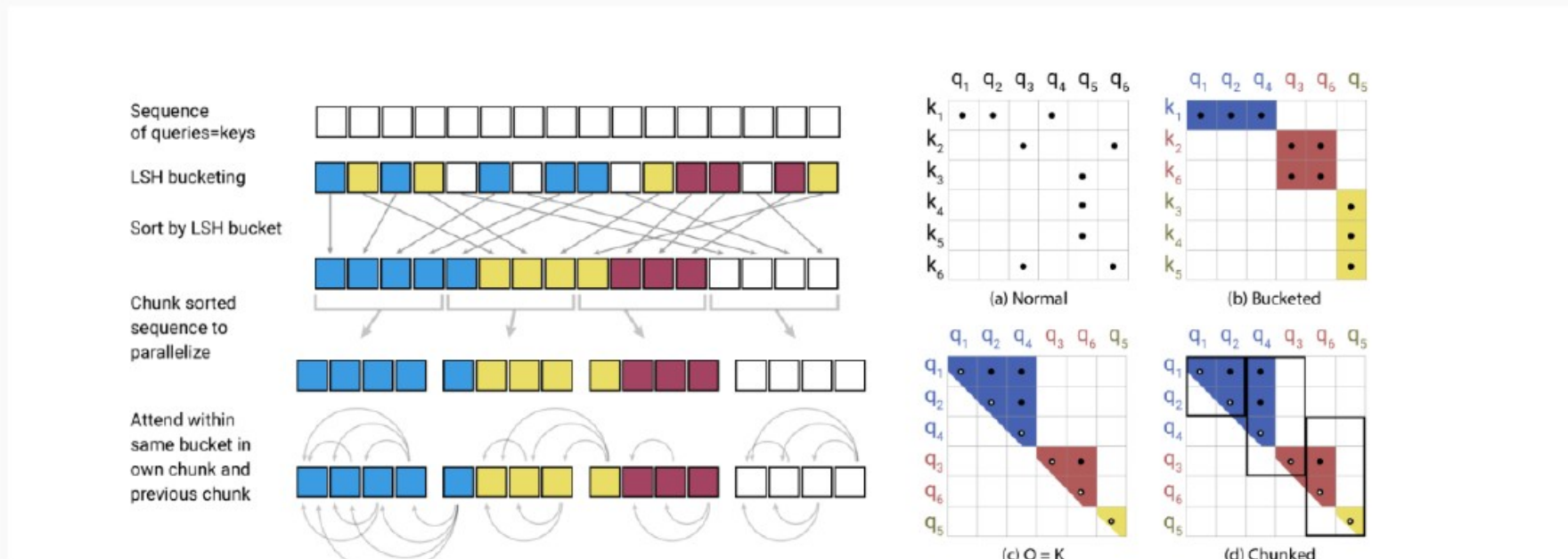
Keep $B = 16384$ buckets and use 5 independent hashing rounds.



A pair succeeds if its full bucket labels agree in at least one round.

Both curves use 100000 trials at each angle.

Sorting brings matching buckets together



- Sort by bucket label and then by original sequence position.
- Split the sorted sequence into chunks of equal length.
- Each query uses its own chunk and the preceding chunk.

A bucket and a chunk serve different purposes

A bucket groups similar directions

Tokens share a hash label. Bucket sizes can vary.

A chunk makes batching practical

Every chunk has m sorted positions. A chunk can cross a bucket boundary.

$$m = \frac{2L}{B}$$

The paper chooses twice the average bucket size. Earlier targets in a bucket fit within two adjacent chunks when the bucket is shorter than m .

Larger buckets can lose useful targets. The method assumes such buckets are rare.

Causal masking follows the original token positions

- Carry the original position indices through sorting.
- Mask targets from the future and targets outside the bucket.
- Penalize self-attention because a query matches its own key most strongly.
- Keep a self-attention fallback when every other target is masked.

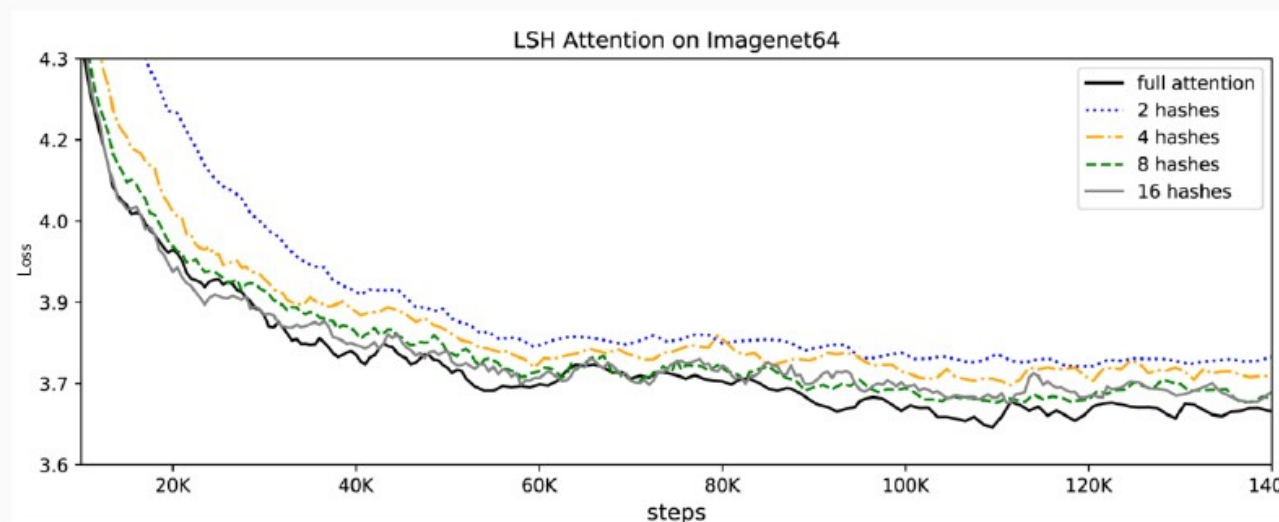
Sorting changes the computation order. The original positions determine which context is allowed.

More hash rounds recover more useful targets

- Each round uses a fresh random projection matrix to form new buckets.
- For each query take the union of valid targets found across rounds.
- A useful target missed in one round can be recovered in another.

More rounds improve coverage and require more computation.

More hashes narrow the ImageNet-64 quality gap



- Eight hashes approach the full-attention curve in this experiment.
- More hashes require more attention computation.
- The curves compare quality after equal numbers of training steps.

Reversible blocks carry two activation streams

$$Y_1 = X_1 + F(X_2)$$

$$Y_2 = X_2 + G(Y_1)$$

- F is the attention block.
- G is the feed-forward block.
- Layer normalization sits inside these functions.
- Each stream has width d_{model} in the paper's comparison.

A scalar network with normal and reversible layers

Both versions take one number x and return one number y .

Normal

$$\begin{aligned}h &\leftarrow x \\h &\leftarrow \text{Transformer}_1(h) \\h &\leftarrow \text{FFN}_1(h) \\h &\leftarrow \text{Convolution}(h) \\h &\leftarrow \text{FFN}_2(h) \\h &\leftarrow \text{Transformer}_2(h) \\h &\leftarrow \text{FFN}_3(h) \\y &\leftarrow h\end{aligned}$$

Reversible

$$\begin{aligned}a &\leftarrow x & b &\leftarrow x \\a &\leftarrow a + \text{Transformer}_1(b) \\b &\leftarrow b + \text{FFN}_1(a) \\a &\leftarrow a + \text{Convolution}(b) \\b &\leftarrow b + \text{FFN}_2(a) \\a &\leftarrow a + \text{Transformer}_2(b) \\b &\leftarrow b + \text{FFN}_3(a) \\y &\leftarrow b\end{aligned}$$

The extra additions change the model. Training keeps the final a and b for reconstruction.

Recover the inputs by subtracting the residuals

$$X_2 = Y_2 - G(Y_1)$$

$$X_1 = Y_1 - F(X_2)$$

1. Start with the two outputs of the block.
2. Recompute $G(Y_1)$ to recover X_2 .
3. Recompute $F(X_2)$ to recover X_1 .

Backpropagation can reconstruct earlier states one block at a time.

Reconstruct the same scalar network in backward pass

Start with the final a and b retained by the forward pass.

$$b \leftarrow b - \text{FFN}_3(a)$$

$$a \leftarrow a - \text{Transformer}_2(b)$$

$$b \leftarrow b - \text{FFN}_2(a)$$

$$a \leftarrow a - \text{Convolution}(b)$$

$$b \leftarrow b - \text{FFN}_1(a)$$

$$a \leftarrow a - \text{Transformer}_1(b)$$

After these six steps both numbers equal the original input x .

Reconstruction replaces storage across depth

Let β be the batch size and N the number of layers.

Usual layer activations

$$O(\beta L d_{\text{model}} N)$$

Keep activations from every layer for the backward pass.

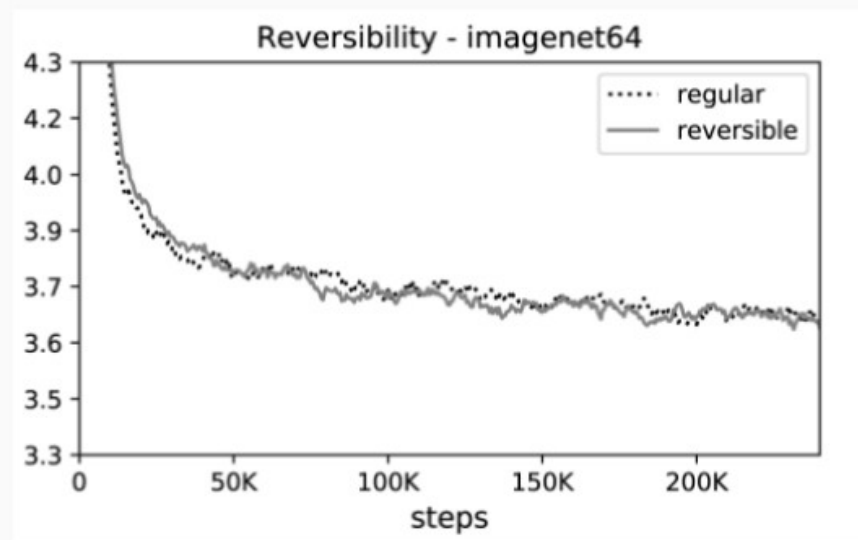
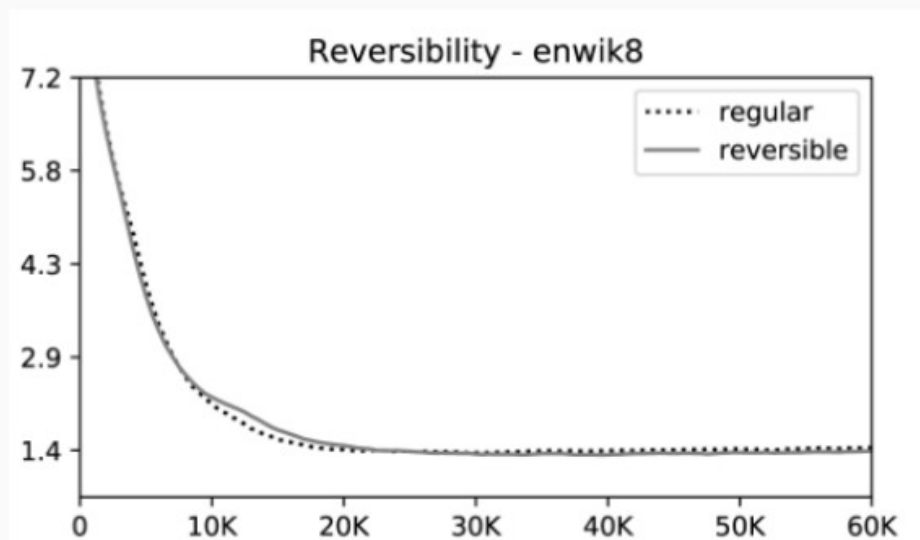
Reversible layer activations

$$O(\beta L d_{\text{model}})$$

Recover earlier states from the current block outputs.

Recomputing the functions takes extra work. Parameter and optimizer storage still grow with model size.

Reversible layers also retain comparable quality



- The comparison uses equal parameter counts and closely matched loss curves.
- Reconstruction still requires extra computation during training.

Translation provides evidence for reversible layers

WMT14 English–German model	BLEU
Original Transformer base after 100K steps	27.3
Reversible base after 100K steps	27.6
Original Transformer big after 300K steps	28.4
Reversible big after 300K steps	29.1

- Higher BLEU is better.
- LSH was omitted because the test sentences fit within a typical 128-token chunk.
- The longer reversible runs untie embedding and output weights.
- Original and reversible models use 100K steps for base and 300K for big.

Feed-forward chunks reduce the wide intermediate

A feed-forward layer applies the same function independently at each position.

$$\underbrace{L \times d_{\text{model}}}_{\text{input}} \longrightarrow \underbrace{L \times d_{\text{ff}}}_{\text{wide intermediate}} \longrightarrow \underbrace{L \times d_{\text{model}}}_{\text{output}}$$

- Process C positions at a time.
- The wide workspace falls from $O(\beta L d_{\text{ff}})$ to $O(\beta C d_{\text{ff}})$.
- Apply the same chunking during reconstruction and backpropagation.

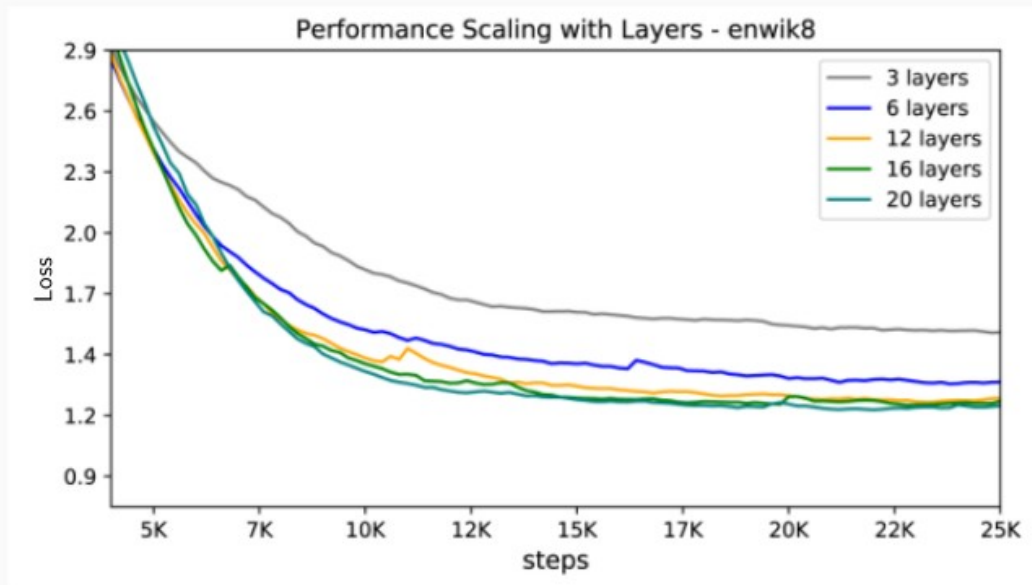
Chunking preserves the position-wise computation. Smaller chunks reduce peak workspace.

Training and evaluation attention must be read together

Training	Attention used at evaluation				
	Full	LSH-8	LSH-4	LSH-2	LSH-1
Full	100%	94.8%	92.5%	76.9%	52.5%
LSH-4	0.8%	100%	99.9%	99.4%	91.9%
LSH-2	0.8%	100%	99.9%	98.1%	86.8%
LSH-1	0.8%	99.9%	99.6%	94.8%	77.9%

- More evaluation hashes help the LSH-trained models.
- Switching those models to full attention reduces accuracy to chance level.
- The paper reports this failure without establishing its cause.

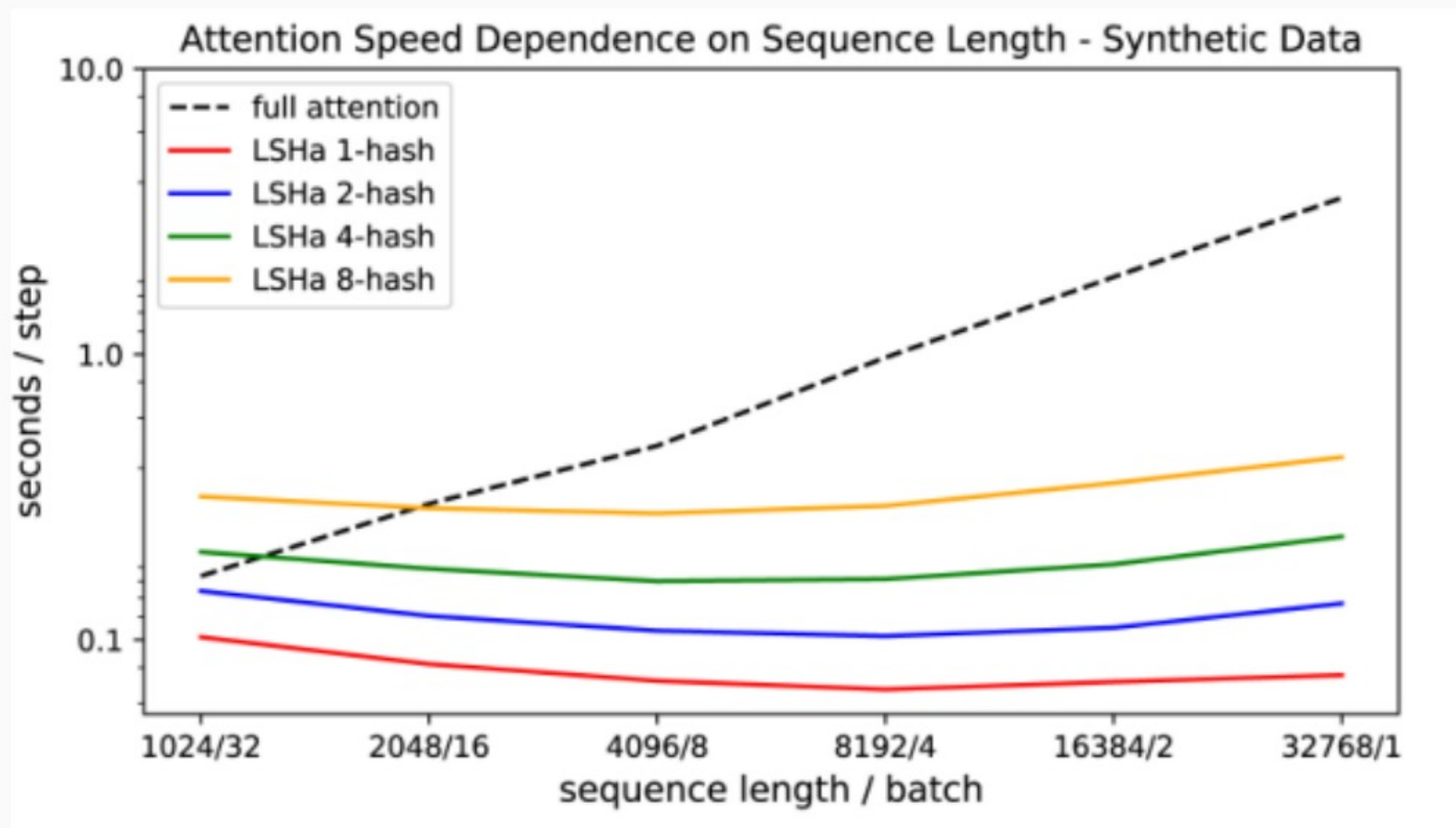
Deeper models improve enwik8 predictions



- Models with up to 20 layers fit and train.
- Deeper models improve over the three-layer model.
- Gains become smaller at greater depth.

A matched deep full-attention baseline was too expensive to run.

Attention evaluation time



Linformer

Self-Attention with Linear Complexity

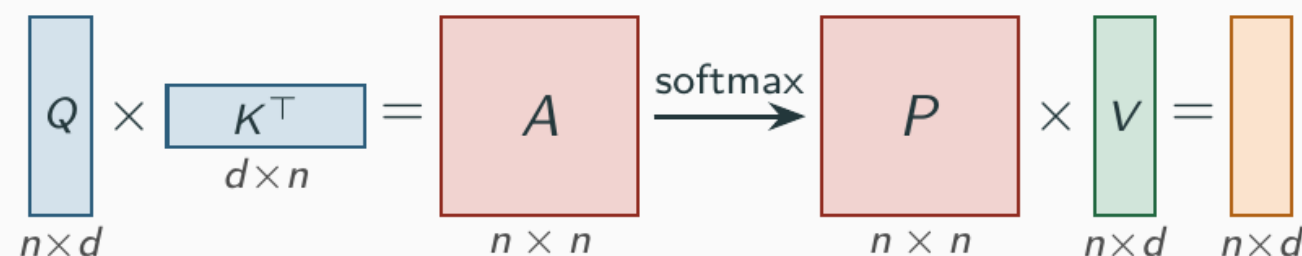
Wang, Li, Khabsa, Fang and Ma, 2020

Presented by Rujula

Self-attention takes quadratic time and memory

One head of attention over n tokens $X \in \mathbb{R}^{n \times d_m}$ computes

$$Q, K, V = XW^Q, XW^K, XW^V \in \mathbb{R}^{n \times d}, \quad P = \text{softmax}(QK^\top / \sqrt{d}), \quad \text{out} = PV$$



Time $O(n^2d)$. Forming $QK^\top$ and PV , in every head of every layer. Doubling n gives $4\times$ the work.

Memory $O(n^2)$. P is kept for the backward pass. In 16-bit floats one P takes 32 MiB at $n = 4096$ and 8 GiB at $n = 65536$.

Linformer replaces the $n \times n$ matrix P with an $n \times k$ one.

The paper asks whether the quadratic step is necessary

The paper answers in three steps.

1. The attention matrix P is approximately low rank. This is measured on a trained model and supported by a theorem.
2. Linformer shortens keys and values from n rows to k rows with learned projections. Time and memory become $O(nk)$.
3. Accuracy stays on par with RoBERTa. At $n = 65536$ and $k = 128$ inference is $20\times$ faster and uses $60\times$ less memory.

We start with what low rank means and why it helps.

A low-rank matrix can be stored as two thin factors

A matrix $P \in \mathbb{R}^{n \times n}$ has rank at most k when it can be written as

$$P = UW^\top \quad U, W \in \mathbb{R}^{n \times k}.$$

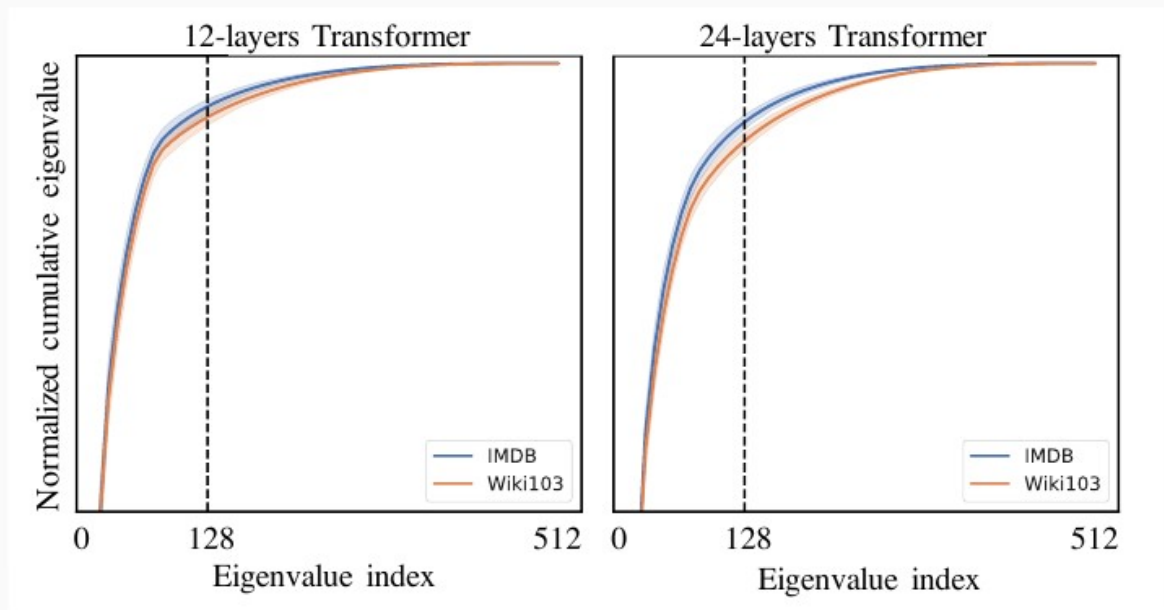
Then the attention output can be computed in a different order.

$$PV = U \underbrace{(W^\top V)}_{k \times d}$$

- Storing U and W takes $2nk$ numbers instead of n^2 .
- The product $W^\top V$ combines the n value rows into k summary rows.
- Each output row is then a mix of these k summaries.

Is trained attention close to low rank?

Trained attention matrices have long-tailed spectra

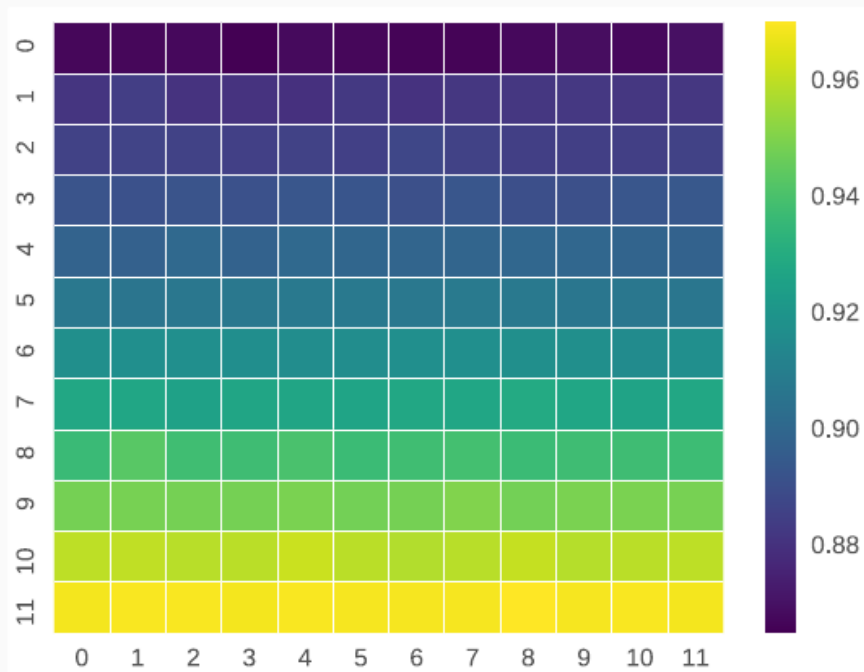


Wang et al. (2020), Fig. 1 left.

- Pretrained RoBERTa-base and RoBERTa-large on Wiki103 and IMDB.
- P is computed for 10k sentences of length $n = 512$, then decomposed by SVD in every layer and head.
- The vertical axis is the share of the first i singular values, $(\sigma_1 + \dots + \sigma_i) / \sum_j \sigma_j$.

The curves rise quickly, so a few directions carry most of P .

Higher layers are closer to low rank



Wang et al. (2020), Fig. 1 right.

- Each cell is one head of RoBERTa-base on Wiki103.
- The colour is the share of the first 128 singular values out of 512.
- Values are high everywhere and highest in the upper layers.

Can low rank be shown in general? Random projections give an argument.

A random projection keeps lengths on average

Draw $R \in \mathbb{R}^{k \times n}$ with $k \leq n$ and independent entries from $\mathcal{N}(0, 1/k)$. Fix a vector $\mathbf{x} \in \mathbb{R}^n$.

$$(R\mathbf{x})_a \sim \mathcal{N}\left(0, \frac{\|\mathbf{x}\|^2}{k}\right) \implies \mathbb{E}\|R\mathbf{x}\|^2 = \|\mathbf{x}\|^2$$

- In matrix form $\mathbb{E}[R^\top R] = I_n$. So $R^\top R$ is a noisy identity of rank k .
- $\|R\mathbf{x}\|^2$ is a sum of k independent terms.
- As k grows the sum concentrates around $\|\mathbf{x}\|^2$.

The Johnson–Lindenstrauss lemma makes “concentrates” precise.

The Johnson–Lindenstrauss lemma bounds the distortion

Lemma, as used in the paper

For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and R as before

$$\Pr(\|R\mathbf{x}\| \leq (1+\epsilon)\|\mathbf{x}\|) > 1 - e^{-(\epsilon^2-\epsilon^3)k/4}$$

$$\Pr(|\mathbf{x}R^\top R\mathbf{y}^\top - \mathbf{xy}^\top| \leq \epsilon |\mathbf{xy}^\top|) > 1 - 2e^{-(\epsilon^2-\epsilon^3)k/4}$$

- The second line says inner products survive the projection.
- The failure probability depends on k and ϵ but not on n .

Row-vector notation as in the paper. Version from Arriaga and Vempala (2006).

Theorem 1 says attention is close to a low-rank matrix

Theorem 1

For any Q, K, V and W^Q, W^K, W^V , and any column $\mathbf{y} \in \mathbb{R}^n$ of VW^V , there is a matrix $\tilde{P} \in \mathbb{R}^{n \times n}$ with

$$\Pr(\|\tilde{P}\mathbf{y} - P\mathbf{y}\| < \epsilon\|P\mathbf{y}\|) > 1 - o(1) \quad \text{and} \quad \text{rank}(\tilde{P}) = \Theta(\log n).$$

- $P\mathbf{y}$ is one column of the attention output, the same feature for all n tokens.
- A matrix of rank $\Theta(\log n)$ reproduces it to relative error ϵ .
- The probability of success tends to one as n grows.

The paper calls the column w . We write $\mathbf{y}$ because the paper reuses w in Theorem 2.

The proof builds $\tilde{P}$ from a random projection.

Proof step 1: build the approximation from a random projection

Draw $R \in \mathbb{R}^{k \times n}$ with independent $\mathcal{N}(0, 1/k)$ entries and set

$$\tilde{P} = P R^\top R.$$

A product has rank at most the rank of each factor, so

$$\text{rank}(\tilde{P}) \leq \text{rank}(R) \leq k.$$

It remains to show that, for $k = \frac{5 \log n}{\epsilon^2 - \epsilon^3}$,

$$\Pr(\|\tilde{P}\mathbf{y} - P\mathbf{y}\| \leq \epsilon \|P\mathbf{y}\|) > 1 - o(1).$$

$$\begin{matrix} \boxed{P} & \boxed{R^\top} & \boxed{R} & = & \boxed{\tilde{P}} \\ n \times n & & & & \text{rank} \leq k \end{matrix}$$

Proof step 2: each output entry is an inner product

Let $\mathbf{p}_i$ be row i of P . Entry i of each output is

$$(P\mathbf{y})_i = \mathbf{p}_i \mathbf{y}, \quad (\tilde{P}\mathbf{y})_i = \mathbf{p}_i R^\top R \mathbf{y}.$$

The row $\mathbf{p}_i$ depends on Q and K but not on R . So the second part of the lemma applies with $\mathbf{x} = \mathbf{p}_i$ and $\mathbf{y}^\top$ in place of $\mathbf{y}$.

Error in one row

$$\Pr \left(|(\tilde{P}\mathbf{y})_i - (P\mathbf{y})_i| > \epsilon |(P\mathbf{y})_i| \right) < 2e^{-(\epsilon^2 - \epsilon^3)k/4}$$

This covers one row. The output has n rows.

Proof step 3: a union bound covers all n rows

The probability that at least one row fails is at most the sum over rows.

$$\Pr(\text{some row fails}) \leq \sum_{i=1}^n \Pr(\text{row } i \text{ fails}) < 2n e^{-(\epsilon^2 - \epsilon^3)k/4}$$

If no row fails, squaring and adding the row bounds gives the bound for the whole column.

$$\|\tilde{P}\mathbf{y} - P\mathbf{y}\|^2 = \sum_{i=1}^n ((\tilde{P}\mathbf{y})_i - (P\mathbf{y})_i)^2 \leq \epsilon^2 \sum_{i=1}^n (P\mathbf{y})_i^2 = \epsilon^2 \|P\mathbf{y}\|^2$$

The failure bound grows with n . Choosing k proportional to $\log n$ cancels this.

Proof step 4: choosing k proportional to $\log n$

Set $k = \frac{5 \ln n}{\epsilon^2 - \epsilon^3}$. Then $e^{-(\epsilon^2 - \epsilon^3)k/4} = n^{-5/4}$ and

$$\Pr(\|\tilde{P}\mathbf{y} - P\mathbf{y}\| \leq \epsilon\|P\mathbf{y}\|) > 1 - 2n \cdot n^{-5/4} = 1 - 2n^{-1/4} = 1 - o(1).$$

The rank of $\tilde{P}$ is at most $k = \Theta(\log n)$. ■

The projection acts along the sequence axis.

$$\tilde{P}V = P R^\top \underbrace{(RV)}_{k \times d}$$

RV has only k rows. Linformer is built on this shape.

Truncated SVD would work but costs too much

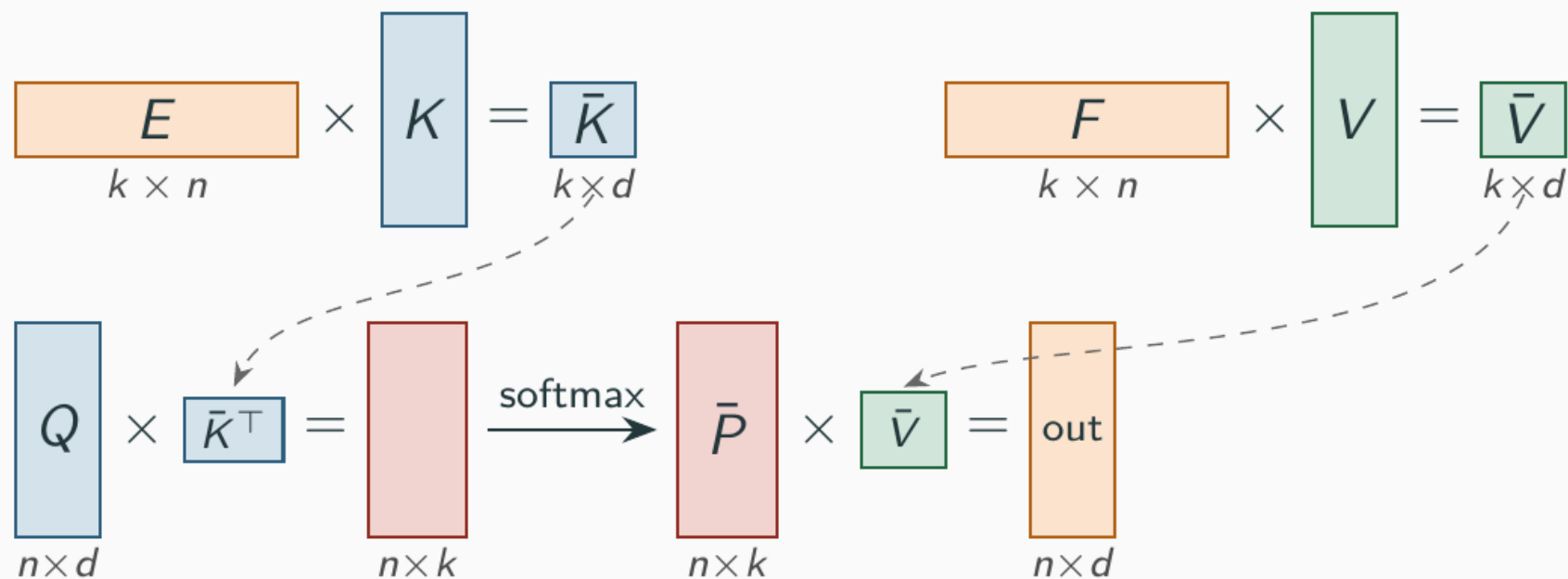
The direct approach keeps the top k singular values of P .

$$P \approx P_{\text{low}} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^{\top}$$

- Truncated SVD gives the best rank- k approximation (Eckart–Young–Mirsky).
- By Theorem 1 this approximates attention with error ϵ in $O(nk)$ time and space.
- But it needs an SVD of every attention matrix, for every head, layer and input.
- The full P must still be formed before it can be decomposed.

Linformer replaces the SVD with learned projections of K and V .

Linformer projects keys and values along the sequence axis



E and F are learned $k \times n$ matrices. They shorten K and V from n rows to k rows. The queries are not projected, so the output still has one row per token.

The largest attention matrix is now $n \times k$ instead of $n \times n$.

Each projected key summarizes all n positions

One Linformer head computes

$$\overline{\text{head}}_i = \underbrace{\text{softmax}\left(\frac{QW_i^Q (E_iKW_i^K)^\top}{\sqrt{d}}\right)}_{\bar{P}: n \times k} \underbrace{F_iVW_i^V}_{k \times d}$$

Row r of the projected keys and values is a weighted sum over every token.

$$\bar{\mathbf{k}}_r = \sum_{j=1}^n E_{rj} \mathbf{k}_j \quad \bar{\mathbf{v}}_r = \sum_{j=1}^n F_{rj} \mathbf{v}_j$$

Each query attends over k summaries instead of n tokens.

Paper eq. 7. The paper writes $E_i, F_i \in \mathbb{R}^{n \times k}$. We write them as $k \times n$ so that $E_iKW_i^K$ is defined.

Linformer is attention with a rank- k mixing matrix

The output can be grouped in two ways.

$$\text{out} = \bar{P} (FV) = (\bar{P}F) V$$

- $\bar{P}F$ is an $n \times n$ matrix that plays the role of P .
- Its rank is at most $\min\{\text{rank } \bar{P}, \text{rank } F\} \leq k$.
- So Linformer is low-rank attention by construction.
- $\bar{P}F$ is never formed. Only $\bar{P}$ ($n \times k$) and FV ($k \times d$) are stored.

Computing in this order is what saves time and memory.

For a fixed k the cost grows linearly with n

Step for one head	Multiply-adds	Memory
$\bar{K} = EK$ and $\bar{V} = FV$	$2nkd$	$2kd$
$S = Q\bar{K}^\top / \sqrt{d}$	nkd	nk
$\bar{P} = \text{softmax}(S)$	$O(nk)$	nk
$\text{out} = \bar{P}\bar{V}$	nkd	nd

Standard attention

$O(n^2d)$ time

$O(n^2 + nd)$ memory

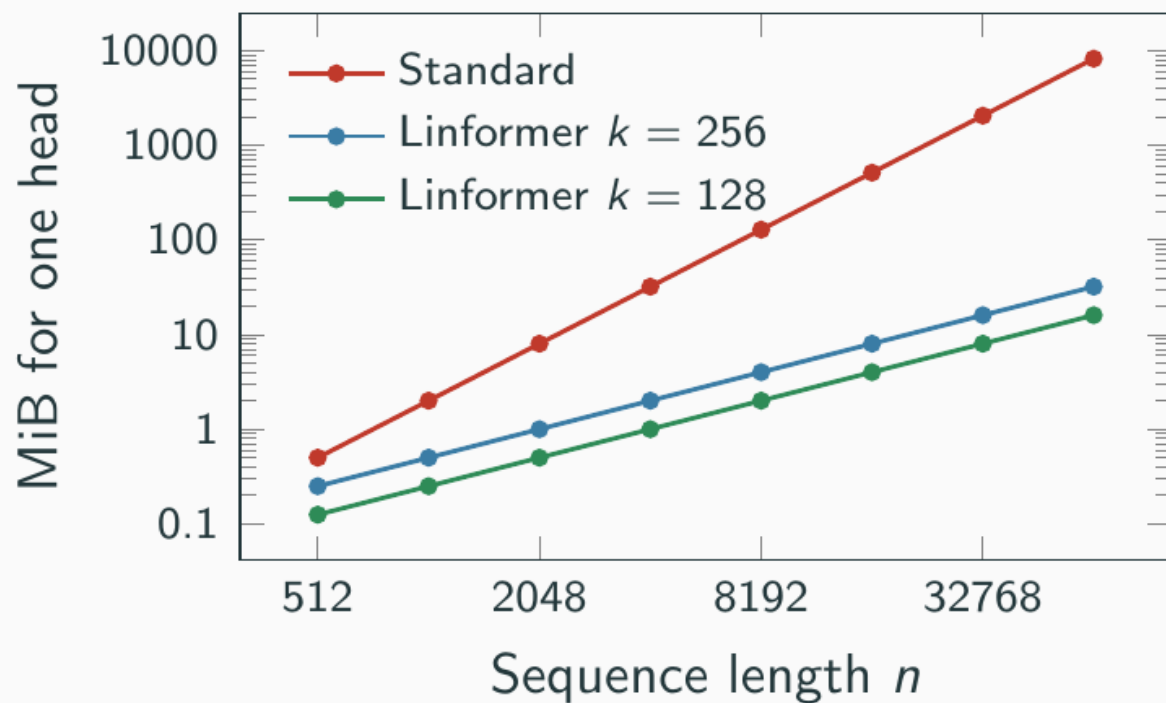
Linformer

$O(nkd)$ time

$O(nk + nd)$ memory

The gap widens as n grows.

The memory gap widens quickly with sequence length



- Scores for one head of one layer in 16-bit floats.
- Standard attention stores n^2 entries.
- Linformer stores nk entries.
- At $n = 65536$ this is 8 GiB against 32 MiB for $k = 256$.

Can k stay fixed while n grows? Theorem 2 addresses this.

Theorem 2 says k need not grow with n

Theorem 2, with E_i, F_i written as $k \times n$

If $k = \min\{\Theta(9d \log(d)/\epsilon^2), 5\Theta(\log(n)/\epsilon^2)\}$, there exist $E_i, F_i \in \mathbb{R}^{k \times n}$ such that, for any row $\mathbf{a}$ of $QW_i^Q(KW_i^K)^\top/\sqrt{d}$,

$$\Pr \left(\left\| \text{softmax}(\mathbf{a}E_i^\top)F_iVW_i^V - \text{softmax}(\mathbf{a})VW_i^V \right\| \leq \epsilon \left\| \text{softmax}(\mathbf{a}) \right\| \left\| VW_i^V \right\| \right) > 1 - o(1).$$

- The first term is Linformer's output for one query. The second is exact attention.
- The bound $\Theta(d \log d/\epsilon^2)$ does not depend on n , so k can stay fixed.
- The proof in the paper's appendix sets $E_i = \delta R$ and $F_i = e^{-\delta} R$. It then uses $\text{rank}(A) = d$ to remove the dependence on n .

The projections can be shared across heads and layers

- *Headwise*. All heads in a layer use the same E and F .
- *Key-value*. Headwise, and also $E = F$.
- *Layerwise*. One matrix E for all layers, heads, keys and values.

12 layers with 12 heads	
Sharing	Projections
none	288
headwise	24
key-value	12
layerwise	1

The paper also proposes a smaller k in higher layers, and pooling or a convolution with stride n/k in place of E . These two ideas are not tested.

The experiments use one value of k in all layers and test the three sharing levels.

Pretraining quality depends on k , not on n

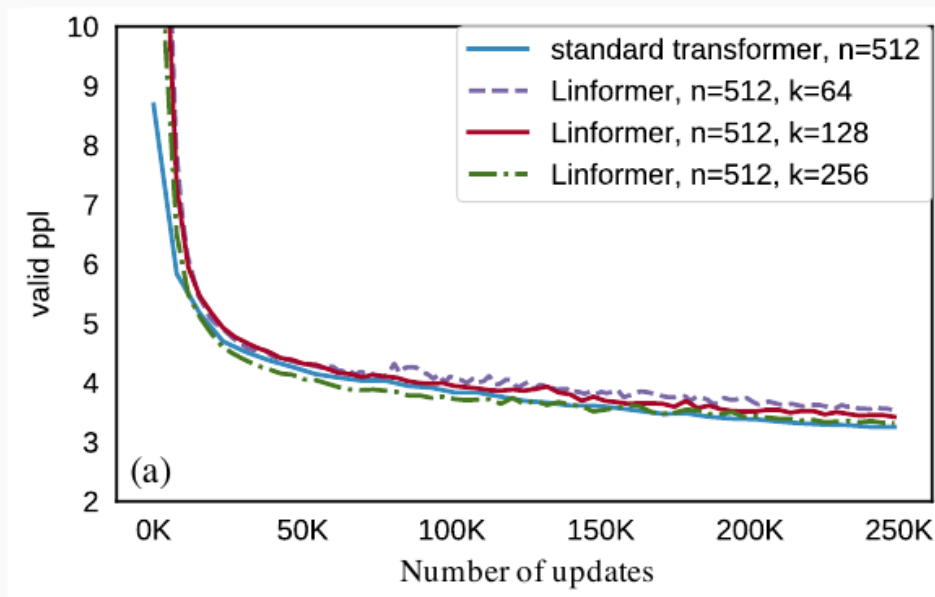


Fig. 3a: $n = 512$, varying k .

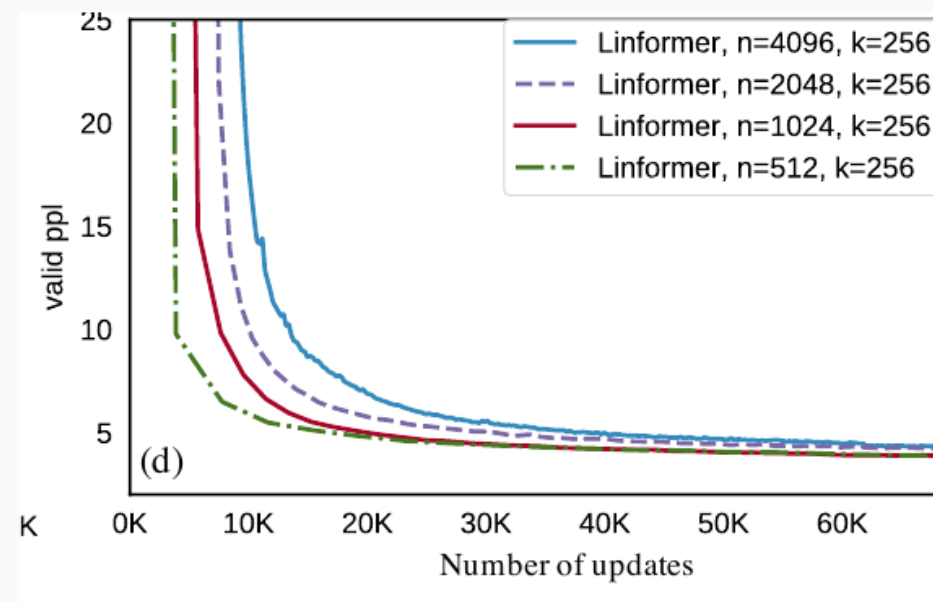


Fig. 3d: $k = 256$, varying n .

Masked language modelling on BookCorpus and English Wikipedia, trained as RoBERTa. Lower perplexity is better.

$k = 128$ is already close to the Transformer, and with k fixed longer sequences do not hurt, as Theorem 2 predicts.

Fine-tuned accuracy is on par with RoBERTa

n	Model	SST-2	IMDB	QNLI	QQP	Average
512	RoBERTa-base	93.1	94.1	90.9	90.9	92.25
512	BERT-base	92.7	93.5	91.8	89.6	91.90
512	DistilBERT	91.3	92.8	89.2	88.5	90.45
512	Linformer, $k = 128$	92.4	94.0	90.4	90.2	91.75
512	Linformer, $k = 256$, layerwise	93.1	94.1	91.2	90.8	92.30
1024	Linformer, $k = 256$, layerwise	93.2	94.2	90.8	90.5	92.18

Wang et al. (2020), Table 2. Development set accuracy. Layerwise means one projection shared by all layers.

Sharing a single projection across all layers costs nothing. Linformer matches RoBERTa and beats DistilBERT.

Speed and memory gains grow with sequence length

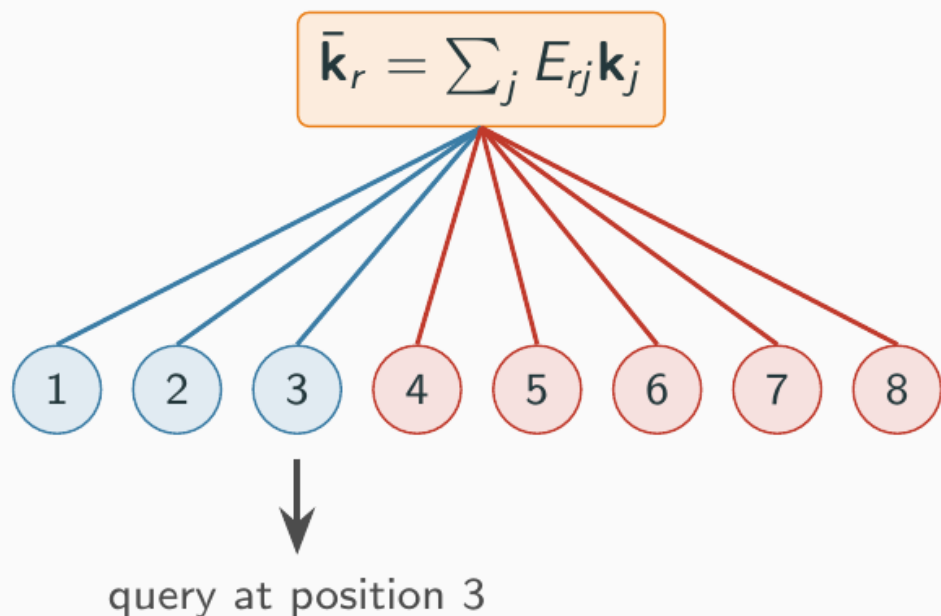
n	Speed-up		Memory saving	
	$k = 128$	$k = 256$	$k = 128$	$k = 256$
512	1.5x	1.3x	1.7x	1.5x
4096	3.4x	3.2x	14x	13x
16384	8.6x	7.8x	56x	48x
65536	20x	18x	60x	52x

Wang et al. (2020), Table 3, selected rows.

- Inference with layerwise-shared Linformer against a Transformer on one 16 GB V100.
- Memory saving is the ratio of the largest batch sizes that fit.

At $n = 65536$ Linformer is $20\times$ faster and uses $60\times$ less memory. Smaller k gives larger gains.

Projected keys mix in future positions



- A decoder must not let token i see tokens after i .
- Standard attention masks those entries of P .
- In Linformer every summary $\bar{\mathbf{k}}_r$ already mixes all n positions, so there is nothing left to mask.

The paper evaluates encoder models only.

Two further limitations

The projections fix a maximum length

E and F have one column per position. They are defined for the maximum sequence length n chosen before training.

Low rank is measured, then assumed

The spectra in Fig. 1 come from pretrained RoBERTa at $n = 512$. Linformer's attention always has rank at most k , so it works well only when the true attention is close to low rank.

Summary

1. Softmax attention builds an $n \times n$ matrix in every head, which costs $O(n^2)$ time and memory.
2. In trained models this matrix is close to low rank, more so in higher layers.
3. Theorem 1 shows that a random projection to $k = \Theta(\log n)$ rows keeps the output close.
4. Linformer learns projections $E, F \in \mathbb{R}^{k \times n}$ for keys and values. Cost becomes $O(nk)$.
5. Theorem 2 shows that k can be chosen independently of n .
6. Accuracy matches RoBERTa at $k = 128$ to 256 , with up to $20\times$ faster inference and $60\times$ less memory at $n = 65536$.

Thank You!

Questions?