Attention Variants

Karl Stratos

Last updated: April, 2026

Contents

1 Linear Attention

Linear attention is best motivated as an effort to make inference recurrent. At each step $t = 1, 2, \ldots$ in test time, a softmax attention layer receives $q_t, k_t, v_t \in \mathbb{R}^d$ and computes

$$o_t = \sum_{i=1}^{t} \text{softmax}_i(K_{1:t} q_t) v_i \tag{1}$$

where $K_{1:t} = (k_1 \ldots k_t) \in \mathbb{R}^{t \times d}$. There is no way to avoid $O(dt)$ compute because the query must ask all t keys at every step. This becomes a runtime bottleneck for long-context generation.[1] But if we skip the softmax, we have

$$o_t = \sum_{i=1}^{t} k_i^\top q_t v_i = S_t q_t \tag{2}$$

where $S_t = \sum_{i=1}^{t} v_i k_i^\top \in \mathbb{R}^{d \times d}$ summarizes the past. Thus we can compute $o_1 \ldots o_T \in \mathbb{R}^d$ recurrently as

$$S_t = S_{t-1} + v_t k_t^\top \qquad\qquad o_t = S_t q_t \tag{3}$$

with $S_0 = 0_{d \times d}$. The total compute is now $O(d^2 T)$ rather than $O(dT^2)$ under softmax.

1.1 Training

(2) admits the usual sequence-batched form: given $Q, K, V \in \mathbb{R}^{T \times d}$,

$$O = \text{lower}(QK^\top)V \in \mathbb{R}^{T \times d} \tag{4}$$

The tradeoff between the recurrent form (3) and the batched form (4) is

1. (3): low compute, but poorly parallelizabile (though see Section 1.1.1)
2. (4): high compute, but well parallelized[2]

But we can combine the two ("chunkwise parallel"). For any segment $1 \le s \le t \le T$ of length $m = t - s + 1$, we can "jump" from S_{s-1} to S_t by

$$S_t = S_{s-1} + V_{s:t}^\top K_{s:t} \tag{5}$$

Writing the t-th output $o_t = S_t q_t$ as an update to S_{s-1}, we have

$$o_t = S_{s-1} q_t + \sum_{i=s}^{t} (q_t^\top k_i) v_i$$

Thus starting from $S_0 \leftarrow 0_{d \times d}$, for each consecutive chunk we can compute the output batch

$$O_{s:t} = \underbrace{Q_{s:t} S_{s-1}^\top}_{\text{inter-chunk}} + \underbrace{\text{lower}(Q_{s:t} K_{s:t}^\top) V_{s:t}}_{\text{intra-chunk}} \in \mathbb{R}^{m \times d} \tag{6}$$

and jump the state by (5).

1.1.1 Prefix-sum parallelization

Even though (3) and (6) have sequential dependencies, it is technically possible to parallelize them by exploiting the fact that they compute prefix sums. Using a parallelized scan algorithm (Appendix A) with $X_t = v_t k_t^\top$

$$S_1 \ldots S_T \leftarrow \textbf{Scan}(X_1 \ldots X_T) \qquad\qquad S_t = \sum_{i=1}^{t} X_i$$

has parallel depth $O(\log T)$ (still with $O(Td^2)$ total work). The same logic applies to chunkwise for length T/m. However, the parallel algorithm tends to be hardware-dependent (e.g., not easily implementable with TPUs) and also has a higher memory overhead $O(Td^2)$ since it materializes all states simultaneously. Thus practical implementations often prefer the simple chunkwise form (6) without a scan.

[1] Even though the compute in (1) is fully parallelizable, the runtime remains $O(t)$ due to memory traffic.

[2] Causal masking is responsible for high compute. If there were no masking, we could compute $O = Q(K^\top V)$ in $O(d^2 T)$ FLOPs.

$J_t(S)$	$R_t(S)$	S_t (argmin of (9))	
$-\langle Sk_t, v_t\rangle$	$\|\|S - S_{t-1}\|\|_F^2$	$S_{t-1} + v_t k_t^\top$	(linear attn)
$-\langle Sk_t, v_t\rangle$	$\|\|S - S_{t-1}\mathrm{diag}\,(\alpha_t)\|\|_F^2$	$S_{t-1}\mathrm{diag}\,(\alpha_t) + v_t k_t^\top$	(GLA)
$\frac{1}{2}\,\|\|Sk_t - v_t\|\|_2^2$	$\|\|S - S_{t-1}\|\|_F^2$	$S_{t-1} - \beta_t(S_{t-1}k_t - v_t)k_t^\top$	(DeltaNet)
$\frac{1}{2}\,\|\|\gamma_t Sk_t - v_t\|\|_2^2$	$\|\|S - \gamma_t S_{t-1}\|\|_F^2$	$\gamma_t S_{t-1} - \beta_t(\gamma_t S_{t-1}k_t - v_t)k_t^\top$	(Gated DN)
$\frac{1}{2}\,\|\|S\mathrm{diag}\,(\alpha_t)\,k_t - v_t\|\|_2^2$	$\|\|S - S_{t-1}\mathrm{diag}\,(\alpha_t)\|\|_F^2$	$S_{t-1}\mathrm{diag}\,(\alpha_t) - \beta_t(S_{t-1}\mathrm{diag}\,(\alpha_t)\,k_t - v_t)k_t^\top$	(KDA)

Table 1: Variational characterization of linear attention variants. We update the state $S \in \mathbb{R}^{d\times d}$ by minimizing a regularized linearization (9) of some per-step loss J_t to promote key-value association, possibly with "forgetting". The memory gates $\alpha_t \in \mathbb{R}^d, \gamma_t \in \mathbb{R}$ and learning rate $\beta_t > 0$ themselves are data-dependent.

1.2 Motivation via Nonparametric Regression

When we write down the attention output at step t

$$o_t = \sum_{i=1}^{t} \frac{\exp(q_t^\top k_i)v_i}{\sum_{j=1}^{t}\exp(q_t^\top k_j)} \tag{7}$$

by pattern matching with (41) we recognize that attention is doing kernel regression (Appendix B) (Tsai *et al.*, 2019). It treats the KV cache $(k_1, v_1)\ldots(k_t, v_t) \in \mathbb{R}^d \times \mathbb{R}^d$ as "labeled data" and predicts an optimal value for the current query $q_t \in \mathbb{R}^d$ with the exponential similarity function $\exp(q^\top k) > 0$.[3] The original linear attention was motivated as a finite-dimensional bilinear approximation $\exp(q^\top k) \approx \phi(q)^\top\phi(k)$ for some QK feature map ϕ (Katharopoulos *et al.*, 2020). Then (7) can be computed recurrently:

$$o_t = \sum_{i=1}^{t} \frac{\phi(q_t)^\top\phi(k_i)v_i}{\sum_{j=1}^{t}\phi(q_t)^\top\phi(k_j)} = \frac{S_t\phi(q_t)}{\phi(q_t)^\top s_t} \qquad S_t = S_{t-1} + v_t\phi(k_t)^\top \in \mathbb{R}^{d\times d} \qquad s_t = s_{t-1} + \phi(k_t) \in \mathbb{R}^d$$

Followup works explored other approximations, such as $\exp(q^\top k) \approx \frac{1}{m}\sum_{r=1}^{m}\phi_{\omega_r}(q)^\top\phi_{\omega_r}(k)$ (i.e. random-feature estimator) in Performer (Choromanski *et al.*, 2020).

2 Linear Attention Variants

2.1 Variational Characterization

A key conceptual leap is that linear attention can be seen as a single step of gradient descent on

$$J_t(S) = -\langle Sk_t, v_t\rangle \tag{8}$$

yielding $S_t = S_{t-1} + v_t k_t^\top$ at $S = S_{t-1}$ with learning rate 1. Thus it is a rare instance where architecture and optimization coincide.[4] This realization is highly profitable because

1. (8) reveals that linear attention optimizes *key-value association*, which is not obvious from the non-recurrent form $o_t = \sum_{i=1}^{t} k_i^\top q_t v_i$.
2. We can consider other objectives or regularizations to promote key-value association "better" and *improve architecture*.

By the usual majorization-minimization principle, given any per-step objective J_t we may minimize its regularized linearization at S_{t-1}

$$S_t = \underset{S\in\mathbb{R}^{d\times d}}{\arg\min}\; J_t(S_{t-1}) + \langle\nabla J_t(S_{t-1}), S - S_{t-1}\rangle + \frac{1}{2\beta_t}R_t(S) \tag{9}$$

Table 1 summarizes some proposed variants.

[3] If $||q||_2 = ||k||_2 = 1$ (e.g., with QKNorm), (7) is the NW estimator using the Gaussian smoothing kernel $\kappa(q,k) = \mathcal{N}(0_d, I_d)(q-k)$.

[4] This concept is appropriately denoted as "fast weights" dating back to the '90s (Schlag *et al.*, 2021). Slow weights (i.e., the learnable parameters of the model) correspond to inter-batch optimization while fast weights (here, the recurrent state) correspond to intra-batch optimization.

2.1.1 Limitations

While the variational characterization is useful for conceptual unification, the variants are independently motivated as pursuing some form of efficient memory management. Crucially, they are developed specifically to enable efficient chunkwise parallel training, thus it is a bit misleading to treat the variational framework as a source of truth. We discuss the three main categories in detail below: (1) delta-rule variants, (2) gated variants, and (3) their hybrids.

2.2 Delta-Rule Variants

The **delta-rule variants** pursue the broad "delta-rule principle" (i.e., learning from error correction) and modify $S_t = S_{t-1} + v_t k_t^\top$ as

$$S_t = S_{t-1} + \beta_t(v_t - \hat{v}_t)k_t^\top \qquad \hat{v}_t = S_{t-1}k_t \tag{10}$$

where β_t is a data-dependent write strength. The idea is that we should only record the residual since memory is precious. It coincides with optimizing $J_t(S) = \frac{1}{2}||Sk_t - v_t||_2^2$ by gradient descent with learning rate β_t (Table 1). To derive a chunkwise parallel form, we can rewrite (10) as a "rank-1 inhomogeneous recurrence"

$$S_t = S_{t-1}(I_d - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top \tag{11}$$

By Corollary C.8, it admits the following chunkwise update.[5] For any segment $1 \le s \le t \le T$ of length $m = t-s+1$, we can jump from S_{s-1} to S_t by

$$S_t = S_{s-1} + (U_{s:t} - W_{s:t}S_{s-1}^\top)^\top K_{s:t} \in \mathbb{R}^{d\times d} \tag{12}$$

where $U_{s:t}, W_{s:t} \in \mathbb{R}^{m\times d}$ are locally computed as

$$U_{s:t} = (I_m + \text{strictLower}(\text{diag}\,(\beta_{s:t})\, K_{s:t}K_{s:t}^\top))^{-1}\text{diag}\,(\beta_{s:t})\, V_{s:t} \tag{13}$$

$$W_{s:t} = (I_m + \text{strictLower}(\text{diag}\,(\beta_{s:t})\, K_{s:t}K_{s:t}^\top))^{-1}\text{diag}\,(\beta_{s:t})\, K_{s:t} \tag{14}$$

Even though this update involves matrix inversion, the inversion of a lower triangular matrix is hardware-efficient. Writing the t-th output $o_t = S_t q_t$ as an update to S_{s-1}, we have from (12)

$$o_t = S_{s-1}\left(q_t - \sum_{i=s}^{t}(q_t^\top k_i)w_i^{(s:t)}\right) + \sum_{i=s}^{t}(q_t^\top k_i)u_i^{(s:t)} \tag{15}$$

Thus starting from $S_0 \leftarrow 0_{d\times d}$, for each consecutive chunk we can compute (13) and (14) locally from $K_{s:t}, V_{s:t} \in \mathbb{R}^{m\times d}$ and $\beta_{s:t} \in \mathbb{R}^m$, compute the output by (15) in batch

$$O_{s:t} = \underbrace{(Q_{s:t} - \text{lower}(Q_{s:t}K_{s:t}^\top)W_{s:t})S_{s-1}^\top}_{\text{inter-chunk}} + \underbrace{\text{lower}(Q_{s:t}K_{s:t}^\top)U_{s:t}}_{\text{intra-chunk}} \in \mathbb{R}^{m\times d} \tag{16}$$

and jump the state by (12).

2.3 Gated Variants

The **gated variants** allow for "forgetting" to make better use of limited memory and modify $S_t = S_{t-1} + v_t k_t^\top$ as

$$S_t = S_{t-1} \odot G_t + v_t k_t^\top \in \mathbb{R}^{d\times d} \tag{17}$$

for some data-dependent forget gate $G_t \in \mathbb{R}^{d\times d}$. This is the same motivation behind LSTMs as gated RNNs.

[5] The derivation uses well-known matrix algebra. For instance, a product of rank-1 updates $I_d - a_i b_i^\top$ (for some $A = (a_1 \dots a_m) \in \mathbb{R}^{m\times d}$ and $B = (b_1 \dots b_m) \in \mathbb{R}^{m\times d}$) can be written as a single low-rank update $I_d - W^\top B$ with recursively defined rows $w_i = a_i - \sum_{j=1}^{i-1}(a_i^\top b_j)w_j$, where the latter furthermore admits the compact expression $W = (I_m + \text{strictLower}(AB^\top))^{-1}A$ (Corollary C.5).

2.3.1 Scalar gating

Let us first focus on the simple scalar gating $G_t = \gamma_t 1_{d\times d}$ for some forget strength $\gamma_t \in (0,1)$ at step t. By unwinding, we have

$$S_t = \gamma_t S_{t-1} + v_t k_t^\top = \sum_{i=1}^{t} \underbrace{\left(\prod_{j=i+1}^{t} \gamma_j\right)}_{r_{i,t}} v_i k_i^\top = r_{s-1,t} S_{s-1} + \sum_{i=s}^{t} r_{i,t} v_i k_i^\top \tag{18}$$

for any segment $1 \le s \le t \le T$ of length $m = t - s + 1$. It is convenient to extract the retention factor $r_{i,t} = c_t / c_i$ as a ratio of global cumprods $c_l = \prod_{i=1}^{l} \gamma_i$. Then we can jump from S_{s-1} to S_t by

$$S_t = \left(\frac{c_t}{c_{s-1}}\right) S_{s-1} + V_{s:t}^\top \mathrm{diag}\left(\frac{c_t}{c_{s:t}}\right) K_{s:t} \tag{19}$$

Writing the t-th output $o_t = S_t q_t$ as an update to S_{s-1}, we have

$$o_t = \left(\frac{c_t}{c_{s-1}}\right) S_{s-1} q_t + \sum_{i=s}^{t} (c_t q_t)^\top \left(\frac{k_i}{c_i}\right) v_i \tag{20}$$

Thus starting from $S_0 \leftarrow 0_{d\times d}$, for each consecutive chunk we can compute the output by (20) in batch

$$O_{s:t} = \mathrm{diag}\left(\frac{c_{s:t}}{c_{s-1}}\right) Q_{s:t} S_{s-1}^\top + \mathrm{lower}((\mathrm{diag}\,(c_{s:t})\, Q_{s:t})(\mathrm{diag}\,(c_{s:t})^{-1} K_{s:t})^\top) V_{s:t} \tag{21}$$

and jump the state by (19).

2.3.2 Columwise gating

GLA (Yang *et al.*, 2023) popularized the form

$$S_t = S_{t-1} \mathrm{diag}\,(\alpha_t) + v_t k_t^\top$$

where $\alpha_t \in \mathbb{R}^d$ gates the state columnwise. This corresponds to scalar gating within individual columns. Let $C \in \mathbb{R}^{T\times d}$ be the elementwise cumprods with rows $C_t = \prod_{i=1}^{t} \alpha_i \in \mathbb{R}^d$. (19) and (20) become[6]

$$S_t \leftarrow S_{s-1} \odot (1_d (C_t / C_{s-1})^\top) + V_{s:t}^\top \left(K_{s:t} \odot \left(1_m C_t^\top \oslash C_{s:t}\right)\right) \tag{22}$$

$$O_{s:t} = (Q_{s:t} \odot (C_{s:t} \oslash 1_m C_{s-1}^\top)) S^\top + \mathrm{lower}((Q_{s:t} \odot C_{s:t})(K_{s:t} \oslash C_{s:t})^\top) V_{s:t} \tag{23}$$

In practice, the cumprods must be computed in log-space for numerical stability which is less hardware-friendly. GLA uses a two-level chunking scheme to improve tensor-core efficiency.

2.4 Hybrid Variants

2.4.1 Scalar-gated DeltaNet

Residual update and gating can be clearly used together. A simple parameterization by Yang *et al.* (2024) is

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

where a data-dependent scalar $\gamma_t \in (0,1)$ gates both the memory and prediction in DeltaNet. This is "almost" a rank-1 inhomogeneous recurrence:

$$S_t = \gamma_t S_{t-1} (I_d - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top \tag{24}$$

[6]Since the scalar multiplication c_t / c_{s-1} for S_{s-1} in (19) now happens for each column, it can be expressed as an elementwise matmul by $1_d (C_t / C_{s-1})^\top \in \mathbb{R}^{d\times d}$. Likewise, the row scaling $c_t / c_{s:t} \in \mathbb{R}^m$ for $K_{s:t} \in \mathbb{R}^{m\times d}$ in (19) now happens for each column, and can be expressed as an elementwise mamul by $1_m C_t^\top \oslash C_{s:t} \in \mathbb{R}^{m\times d}$. The row scaling $c_{s:t} / c_{s-1}$ in (21) similarly becomes an elementwise matmul by $C_{s:t} \oslash 1_m C_{s-1}^\top \in \mathbb{R}^{m\times d}$.

The key reparameterization trick is to divide by the cumprod $c_t = \prod_{i=1}^t \gamma_i$. Let $\bar{S}_t = S_t/c_t$. Using the fact that $c_t = \gamma_t c_{t-1}$, in particular $\gamma_t S_{t-1} = c_t \bar{S}_{t-1}$, we have

$$\bar{S}_t = \bar{S}_{t-1}(I_d - \beta_t k_t k_t^\top) + \frac{\beta_t}{c_t} v_t k_t^\top \tag{25}$$

Thus by Corollary C.8 it admits the same chunkwise update form (12) $\bar{S}_t = \bar{S}_{s-1} + (\bar{U}_{s:t} - W_{s:t}\bar{S}_{s-1}^\top)^\top K_{s:t}$ except that $\bar{U}_{s:t}$ uses $\operatorname{diag}(\beta_{s:t}/c_{s:t})\, V_{s:t}$ instead of $\operatorname{diag}(\beta_{s:t})\, V_{s:t}$ in (13). Since $o_t = S_t q_t = c_t \bar{S}_t q_t$, from (16) we have $O_{s:t} = \operatorname{diag}(c_{s:t})\, \bar{O}_{s:t}$ where $\bar{O}_{s:t}$ is computed as in (16) except that it uses $\bar{S}_{s-1}$ and $\bar{U}_{s:t}$.

2.4.2 Columnwise-gated DeltaNet

We can also combine GLA with DeltaNet as in KDA (Kimi Team, 2025):

$$S_t = S_{t-1}\operatorname{diag}(\alpha_t) - \beta_t \left(S_{t-1}\operatorname{diag}(\alpha_t)\, k_t - v_t\right) k_t^\top$$

which is an exact analog of scalar-gated DeltaNet for columnwise gating by data-dependent $\alpha_t \in (0,1)^d$. Again, this is almost a rank-1 inhomogenous recurrence:

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

where we shorthand $D_t = \operatorname{diag}(\alpha_t)$. Again, the key reparameterization trick is $\widetilde{S}_t = S_t C_t^{-1}$ with the cumprod $C_t = \prod_{i=1}^t D_t$. Using the fact that $C_t = C_{t-1} D_t$, in particular $S_{t-1} D_t = \widetilde{S}_{t-1} C_t$, we have

$$\widetilde{S}_t = \widetilde{S}_{t-1}\left(I_d - \beta_t (C_t k_t)(C_t^{-1} k_t)^\top\right) + \beta_t v_t (C_t^{-1} k_t)^\top$$

Thus by Corollary C.8 it admits the chunkwise jump

$$\widetilde{S}_t = \widetilde{S}_{s-1} + (\widetilde{U}_{s:t} - \widetilde{W}_{s:t}\widetilde{S}_{s-1}^\top)^\top (K_{s:t} \oslash C_{s:t}) \tag{26}$$

where $C_{s:t} \in \mathbb{R}^{m \times d}$ has rows $C_s \dots C_t \in \mathbb{R}^d$ (un-diag-ed) which can be computed chunkwise, with locally computed

$$\widetilde{U}_{s:t} = (I_m + \operatorname{strictLower}(\operatorname{diag}(\beta_{s:t})\,(K_{s:t} \odot C_{s:t})(K_{s:t} \oslash C_{s:t})^\top))^{-1} \operatorname{diag}(\beta_{s:t})\, V_{s:t} \tag{27}$$

$$\widetilde{W}_{s:t} = (I_m + \operatorname{strictLower}(\operatorname{diag}(\beta_{s:t})\,(K_{s:t} \odot C_{s:t})(K_{s:t} \oslash C_{s:t})^\top))^{-1} \operatorname{diag}(\beta_{s:t})\,(K_{s:t} \odot C_{s:t}) \tag{28}$$

Since $o_t = S_t q_t = \widetilde{S}_t C_t q_t$, we have

$$O_{s:t} = \underbrace{(Q_{s:t} \odot C_{s:t} - \operatorname{lower}((Q_{s:t} \odot C_{s:t})(K_{s:t} \oslash C_{s:t})^\top)\widetilde{W}_{s:t})\widetilde{S}_{s-1}^\top}_{\text{inter-chunk}} + \underbrace{\operatorname{lower}((Q_{s:t} \odot C_{s:t})(K_{s:t} \oslash C_{s:t})^\top)\widetilde{U}_{s:t}}_{\text{intra-chunk}} \tag{29}$$

3 Data-Dependent Relative Position Encoding

A general form of relative position encoding is

$$o_t = \frac{\sum_{i=1}^t \exp\left(q_t^\top H_{t,i} k_i + b_{t,i}\right) v_i}{\sum_{l=1}^t \exp\left(q_t^\top H_{t,l} k_l + b_{t,l}\right)} \tag{30}$$

where $H_{t,i} \in \mathbb{R}^{d \times d}$ and $b_{t,i} \in \mathbb{R}$ model the relative positions $t \geq i$ (with $H_{t,t} = I_d$ and $b_{t,t} = 0$). The standard RoPE attention defines $H_{t,i}$ as a data-independent block-diagonal matrix where the j-th block

$$H_{t,i}^{(j)} = \begin{bmatrix} \cos((i-t)\theta_j) & -\sin((i-t)\theta_j) \\ \sin((i-t)\theta_j) & \cos((i-t)\theta_j) \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

is a relative rotation by $(i-t)\theta_j$ radians with a geometrically decaying base degree $\theta_j = (10000)^{-2(j-1)/d}$. Recent work propose to make $H_{t,i}, b_{t,i}$ data-dependent drawing inspiration from linear attention variants. Again, the crucial practical consideration is maintaining efficient training, specifically by a FlashAttention-style kernel that can process $O(T^2)$ query-key scores in tiles.

3.1 FoX

FoX (Lin *et al.*, 2025) proposes the parameterization

$$o_t = \frac{\sum_{i=1}^{t} \exp\left(q_t^\top k_i + \sum_{j=i+1}^{t} \log \gamma_j\right) v_i}{\sum_{i=1}^{t} \exp\left(q_t^\top k_i + \sum_{j=i+1}^{t} \log \gamma_j\right)} \qquad \gamma_j = \sigma(w_{\text{fox}}^\top h_j + b_{\text{fox}}) \tag{31}$$

Here, we may view $\gamma_t \in (0,1)$ as the same kind of forget strength in scalar-gated linear attention $S_t = \gamma_t S_{t-1} + v_t k_t^\top$ (Section 2.3.1) by "restoring" the exponential similarity function $q_t^\top k_i \Rightarrow \exp(q_t^\top k_i)$:

$$o_t = \sum_{i=1}^{t} \left(\prod_{j=i+1}^{t} \gamma_j \right) q_t^\top k_i v_i \qquad \Rightarrow \qquad o_t = \sum_{i=1}^{t} \exp\left(q_t^\top k_i + \sum_{j=i+1}^{t} \log \gamma_j \right) v_i$$

where the RHS becomes (31) when normalized.

3.1.1 Training

FoX is especially convenient to implement because it does not disrupt tiling. It admits the forward form $O = \text{softmax}(QK^\top + D)V$ where $D = c1_T^\top - 1_T c^\top$ can be computed using the cumsumlog $c = \text{cumsum}(\log \gamma)$. Thus we may pass $c \in \mathbb{R}^T$ as an additional input to the FlashAttention kernel, which then computes each local QK product as $Q_I K_J^\top + c_I 1_C^\top - 1_C c_J^\top$ and proceeds as usual.

3.2 PaTH

PaTH (Yang *et al.*, 2025) proposes the parameterization

$$o_t = \frac{\sum_{i=1}^{t} \exp\left(k_i^\top \prod_{j=i+1}^{t} (I_d - \beta_j \nu_j \nu_j^\top) q_t\right) v_i}{\sum_{i=1}^{t} \exp\left(k_i^\top \prod_{j=i+1}^{t} (I_d - \beta_j \nu_j \nu_j^\top) q_t\right)} \qquad \beta_j = 2\sigma(w_{\text{path}}^\top h_j + b_{\text{path}}) \qquad \nu_j = \text{LightLayer}(h_j) \tag{32}$$

The choice of $H_{t,i}^\top = \prod_{j=i+1}^{t} M_j$ (note the transpose to align the "direction from key to query") as a product of rank-1 updates $M_j = I_d - \beta_j \nu_j \nu_j^\top$ can be motivated as emulating the DeltaNet state evolution which takes the form of $S_t = S_{t-1} M_t + X_t = \sum_{i=1}^{t} X_i H_{t,i}$ for $X_i \propto v_t k_t^\top$ (11). Then restoring the expoential similarity in DeltaNet as

$$o_t \approx \sum_{i=1}^{t} q_t^\top H_{t,i} k_i v_i \qquad \Rightarrow \qquad o_t = \sum_{i=1}^{t} \exp\left(q_t^\top H_{t,i} k_i\right) v_i$$

we obtain (32) by normalizing.

3.2.1 Tiling algorithm

PaTH only modifies the attention score as

$$A_{t,i}^{\text{path}} = k_i^\top P_{i+1:t} q_t \qquad \forall 1 \le i \le t \le T \tag{33}$$

where $P_{i+1:t} := \prod_{j=i+1}^{t} M_j \in \mathbb{R}^{d \times d}$ is seen as a key-to-query "path". To derive tiling, suppose k_i and q_t live in intervals $I = [*, b_{\text{key}}]$ and $J = [a_{\text{query}}, *]$. Then (33) can be decomposed as

$$A_{t,i}^{\text{path}} = \begin{cases} k_i^\top \left(\prod_{j=i+1}^{t} M_j \right) q_t & (\text{if } I = J) \\ \left(k_i^\top \left(\prod_{j=i+1}^{b_{\text{key}}} M_j \right) \right) \left(\prod_{[a,b] \in \mathcal{C}_{\text{inter}}} P_{a:b} \right) \left(\left(\prod_{j=a_{\text{query}}}^{t} M_j \right) q_t \right) & (\text{otherwise}) \end{cases} \tag{34}$$

where $\mathcal{C}_{\text{inter}}$ denote the (ordered) intermediate intervals. Suppose that we can calculate for every interval I

- $A_{I,I}^{\text{path}} \in \mathbb{R}^{|I| \times |I|}$: $(A_{I,I}^{\text{path}})_{\text{loc}_I(t), \text{loc}_I(i)} = k_i^\top \left(\prod_{j=i+1}^{t} M_j \right) q_t$ for all $i \le t$ in I
- $\overleftarrow{Q}_I \in \mathbb{R}^{|I| \times d}$: $(\overleftarrow{Q}_I)_{\text{loc}_I(t)} = (\prod_{j=I_1}^{t} M_j) q_t$ for all $t \in I$

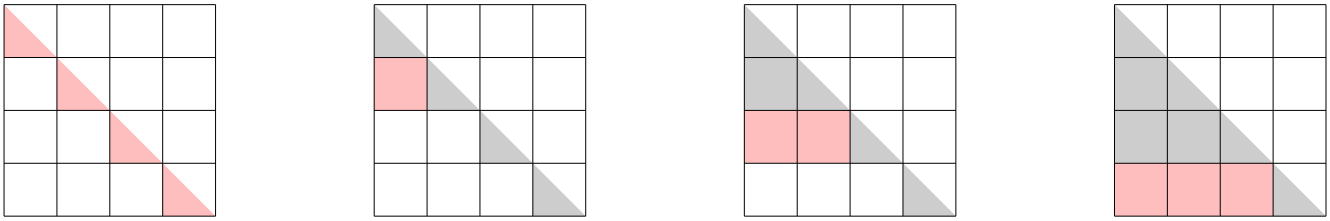


Figure 1: Illustration of the tiling algorithm for PaTH.

- $\overrightarrow{K}_I \in \mathbb{R}^{|I|\times d}$: $(\overrightarrow{K}_I)_{\mathrm{loc}_I(i)} = (\prod_{j=i+1}^{I_2} M_j)^\top k_i$ for all $i \in I$

where $\mathrm{loc}_I(i) = i - I_1 + 1$ translates $i \in I$ to local indices $\{1 \dots |I|\}$. Then we can fill out $A^{\text{path}} \in \mathbb{R}^{T\times T}$ in tiles as

- For each I, precompute $A^{\text{path}}_{I,I}$, $\overleftarrow{Q}_I$, $\overrightarrow{K}_I$, and initialize $\overleftarrow{Q}_{I,\text{inter}} \leftarrow \overleftarrow{Q}_I$.
- For each I, for each $J < I$ (right-to-left), compute $A^{\text{path}}_{I,J} \leftarrow \overleftarrow{Q}_{I,\text{inter}} \overrightarrow{K}_J^\top$ and update $\overleftarrow{Q}_{I,\text{inter}} \leftarrow \overleftarrow{Q}_{I,\text{inter}} P_J^\top$.

The correctness of the algorithm is evident from (34). See Figure 1 for illustration.

3.2.2 Derivation of causal masking

Causal masking is needed to compute $A^{\text{path}}_{I,I}$, $\overleftarrow{Q}_I$, and $\overrightarrow{K}_I$ without looping. Since a local path $P_{s:t} = \prod_{i=s}^{t}(I_d - \beta_i \nu_i \nu_i^\top)$ is a product of $m \geq 1$ rank-1 updates, it admits a compact expression via a triangular solve (Corollary C.5)

$$P_{s:t} = I_d - ((I_m + \text{strictLower}(\text{diag}\,(\beta_{s:t})\, N_{s:t} N_{s:t}^\top))^{-1} \text{diag}\,(\beta_{s:t})\, N_{s:t})^\top N_{s:t} \tag{35}$$

However, this alone does not imply any parallel form since each $[s,t]$ seems to require its own inverse. It turns out this is not the case! One way to see this is to take an explicity DAG view of (35) as computing a *strictly lower triangular* and *pairwise-local* adjacency matrix over m nodes by $E^{(s:t)} = -\text{strictLower}(\text{diag}\,(\beta_{s:t})\, N_{s:t} N_{s:t}^\top)$ and their all-path-sums by $R^{(s:t)} = (I_m - E^{(s:t)})^{-1}$ (Corollary C.2). Thanks to the monotonicity of this DAG, we can obtain $R^{(s:t)} \in \mathbb{R}^{m\times m}$ by taking the corresponding principal submatrix of any all-path-sum matrix $R_I \in \mathbb{R}^{|I|\times|I|}$ such that $[s,t] \subseteq I$ (Corollary C.11). To avoid complicated indexing, we will work with the global $R = R_{1:T}$ for the derivation. Let $\Lambda = R\text{diag}\,(\beta) \in \mathbb{R}^{T\times T}$ where each entry $\Lambda_{i,j} \in \mathbb{R}$ computes the all-path-sum from i to j multiplied by an "end scale" $\beta_j \in \mathbb{R}$. Then (35) admits the quadratic masked form

$$P_{s:t} = I_d - (\text{diag}\,(1_{s:t})\, N)^\top \Lambda^\top (\text{diag}\,(1_{s:t})\, N) \tag{36}$$

where $1_{s:t} \in \{0,1\}^T$ preserves only $N_{s:t}$. Then the PaTH score (33) becomes

$$\begin{aligned} A^{\text{path}}_{t,i} = k_i^\top P_{i+1:t} q_t &= q_t^\top k_i - q_t (\text{diag}\,(1_{i+1:t})\, N)^\top \Lambda (\text{diag}\,(1_{i+1:t})\, N) k_i \\ &= q_t^\top k_i - \sum_{i<p\leq l\leq t} \Lambda_{l,p} \times (q_t^\top \nu_l) \times (k_i^\top \nu_p) \qquad \text{(since } \Lambda \text{ is lower triangular)} \end{aligned}$$

for all $t \geq i$. This implies the global causal masking

$$\begin{aligned} A^{\text{path}} &= \text{lower}(QK^\top) - \text{lower}(QN^\top)\, \Lambda\, \text{strictLower}(NK^\top) \in \mathbb{R}^{T\times T} \\ \overleftarrow{Q} &= Q - \text{lower}(QN^\top)\, \Lambda\, N \in \mathbb{R}^{T\times d} \\ \overrightarrow{K} &= K - \text{strictLower}(NK^\top)^\top\, \Lambda^\top\, N \in \mathbb{R}^{T\times d} \end{aligned}$$

Finally, tiled causal masking immediately follows from the fact that $\Lambda_{I:I} = \Lambda^{(I)}$ where $\Lambda^{(I)} = R^{(I)}\text{diag}\,(\beta_I) \in \mathbb{R}^{|I|\times|I|}$ is locally computed—again because Λ is an all-path-sum (times an end scale) matrix which is is closed under principal submatrix selection. Thus

$$\begin{aligned} A^{\text{path}}_{I,I} &= \text{lower}(Q_I K_I^\top) - \text{lower}(Q_I N_I^\top)\, \Lambda^{(I)}\, \text{strictLower}(N_I K_I^\top) \in \mathbb{R}^{|I|\times|I|} \\ \overleftarrow{Q}_I &= Q_I - \text{lower}(Q_I N_I^\top)\, \Lambda^{(I)}\, N_I \in \mathbb{R}^{|I|\times d} \\ \overrightarrow{K}_I &= K_I - \text{strictLower}(N_I K_I^\top)^\top\, (\Lambda^{(I)})^\top\, N_I \in \mathbb{R}^{|I|\times d} \end{aligned}$$

References

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., *et al.* (2020). Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

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

Kimi Team (2025). Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*.

Lin, Z., Nikishin, E., He, X. O., and Courville, A. (2025). Forgetting transformer: Softmax attention with a forget gate. *arXiv preprint arXiv:2503.02130*.

Schlag, I., Irie, K., and Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. (2019). Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 4344–4353.

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

Yang, S., Kautz, J., and Hatamizadeh, A. (2024). Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*.

Yang, S., Shen, Y., Wen, K., Tan, S., Mishra, M., Ren, L., Panda, R., and Kim, Y. (2025). Path attention: Position encoding via accumulating householder transformations. *arXiv preprint arXiv:2505.16381*.

A Scan

Let $\mathcal{X}$ be an additive group (i.e., things can be "added" and there is a "0").[7] Let $X_1 \dots X_T \in \mathcal{X}$ where T is a power of 2 for convenience (otherwise pad with 0s). Thanks to associativity, we can trivially build a binary tree where each node z computes the sum of its span by $\mathbf{Sum}(z) = \mathbf{Sum}(z.\texttt{left}) + \mathbf{Sum}(z.\texttt{right})$, in particular computing the total sum $\mathbf{Sum}(z_{\texttt{root}}) = \sum_{t=1}^{T} X_t$ in $O(\log T)$ time if implemented with hierarchical parallelism (the total work remains $O(T)$).

Now, the **scan** of $X_1 \dots X_T$ refers to *all* prefix sums $P_1 \dots P_T \in \mathcal{X}$ where $P_t = \sum_{i \le t} X_i$. We can compute them in parallel by the following insight. Let $\mathbf{Sum}_{\leftarrow}(z)$ denote the sum of the leaves of the "outside tree" rooted at z constrained to the left side. Then

1. When z corresponds to a leaf X_t, then $\mathbf{Sum}_{\leftarrow}(z) = \sum_{i<t} X_i$ (i.e., we can get P_t from this by adding X_t).

2. At any node z, we have

$$\begin{aligned}\mathbf{Sum}_{\leftarrow}(z.\texttt{left}) &= \mathbf{Sum}_{\leftarrow}(z)\\ \mathbf{Sum}_{\leftarrow}(z.\texttt{right}) &= \mathbf{Sum}_{\leftarrow}(z) + \mathbf{Sum}(z.\texttt{left})\end{aligned}$$

This yields a two-pass parallel scan algorithm where we compute $\mathbf{Sum}(z)$ for all nodes z bottom-up, then compute $\mathbf{Sum}_{\leftarrow}(z)$ for all nodes z top-down. This preserves $O(T)$ work and $O(\log T)$ depth.

[7] This is technically stronger than needed since we do not require an inverse, but we will not go to that level of pedanticity.

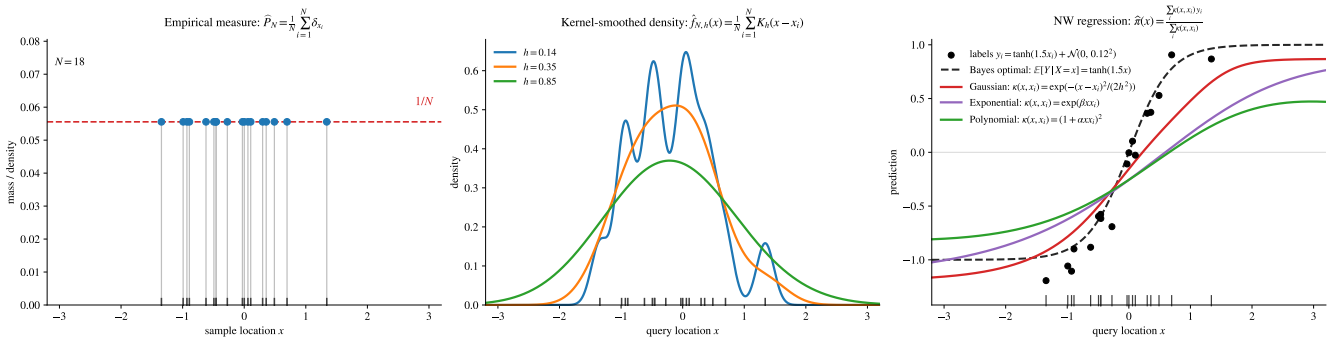


Figure 2: (Left) Empirical measure. (Middle) Kernel-smoothed density using the Gaussian kernel $K(z) = \mathcal{N}(0,1)(z)$ with various bandwidths. (Right) The NW estimator $\mathbf{E}[Y|X=x] \approx \frac{\sum_{i=1}^N \kappa(x,x_i) y_i}{\sum_{i=1}^N \kappa(x,x_i)}$ with various similarity functions.

B Kernel Smoothing

We often estimate density $\hat{f} \approx f_X$ given N iid samples $x_1 \dots x_N \in \mathbb{R}^d$ of $X \sim f_X$. The empirical measure is

$$\widehat{P}_N = \frac{1}{N}\sum_{i=1}^{N} \delta_{x_i} \tag{37}$$

where $\delta_{x_i}(S) = \mathbb{1}(x_i \in S)$ is a Dirac measure.[8] **Kernel density estimation (KDE)** smooths (37) into a density estimate

$$\hat{f}_N(x) = \mathop{\mathbf{E}}_{u \sim \widehat{P}_N}[K(x-u)] = \frac{1}{N}\sum_{i=1}^{N} K(x - x_i) \tag{38}$$

which is a proper density if $K : \mathbb{R}^d \to \mathbb{R}$ is a density (aka. **smoothing kernel**); taking K centered at 0 makes it a sensible local smoother around each sample. To control the degree of approximation, we introduce a bandwidth parameter $h > 0$ and use $K_h(z) = (1/h^d)K(z/h)$ (the normalization ensures $\int_z K_h(z)dz = 1$). As $h \to 0$, $K_h(x - x_i)$ shrinks to a point mass at x_i, so the measure $\hat{f}_{N,h}(x)dx$ converges to (37) (Figure 2 middle).

B.1 Application to Nonparametric Regression

The goal of regression $X \mapsto Y \in \mathbb{R}^p$ is to make Bayes optimal prediction $\pi^\star(x) = \mathbf{E}[Y|X=x]$. Kernel smoothing allows us to directly estimate $\pi^\star(x)$ in a nonparametric fashion. Given $(x_1, y_1) \dots (x_N, y_N) \sim f_{XY}$, we use KDE

$$\hat{f}_{N,h}(x,y) = \frac{1}{N}\sum_{i=1}^{N} K_h(x - x_i) L_h(y - y_i) \qquad \hat{f}_{N,h}(x) = \frac{1}{N}\sum_{i=1}^{N} K_h(x - x_i)$$

where $K_h : \mathbb{R}^d \to \mathbb{R}$ and $L_h : \mathbb{R}^p \to \mathbb{R}$ are some smoothing kernels with bandwidth $h > 0$. Then

$$\pi^\star(x) = \int_y \frac{f_{XY}(x,y)y}{f_X(x)} dy \approx \int_y \frac{\hat{f}_{N,h}(x,y)y}{\hat{f}_{N,h}(x)} dy = \frac{\sum_i K_h(x - x_i) \int_y L_h(y - y_i) y dy}{\sum_i K_h(x - x_i)} = \frac{\sum_i K_h(x - x_i) y_i}{\sum_i K_h(x - x_i)} =: \hat{\pi}(x) \tag{39}$$

(the penultimate step uses the fact that L_h is a centered density) where $\hat{\pi}$ is known as the **Nadaraya-Watson estimator**. It is shown to be consistent for bandwidths satisfying $h \to 0$ and $Nh^d \to \infty$ as a function of N.

B.1.1 Broader usage

We make two observations about the NW estimator (39):

$$\hat{\pi}(x) = \frac{\sum_{i=1}^N K_h(x - x_i) y_i}{\sum_{i=1}^N K_h(x - x_i)}$$

[8] The indicator formulation $\hat{p}(x) = \frac{1}{N}\sum_{i=1}^N \mathbb{1}(x = x_i)$ is not a proper density since $\int \hat{p}(x)dx = 0$.

1. Smoothing kernel $K_h : \mathbb{R}^d \to \mathbb{R}$ does not need normalization (unlike KDE).
2. Variationally, $\hat{\pi}$ can be expressed as performing weighted regression against observed data (easily checked):

$$\hat{\pi}(x) = \arg\min_{y \in \mathbb{R}^p} \sum_{i=1}^{N} K_h(x - x_i) \, ||y - y_i||_2^2 \tag{40}$$

This lets us loosely talk about nonparametric regression as weighted average, taking any positive similarity function $\kappa(x, x') > 0$ and predicting

$$\tilde{\pi}(x) = \arg\min_{y \in \mathbb{R}^p} \sum_{i=1}^{N} \kappa(x, x_i) \, ||y - y_i||_2^2 = \frac{\sum_{i=1}^{N} \kappa(x, x_i) y_i}{\sum_{i=1}^{N} \kappa(x, x_i)} \tag{41}$$

which coincides with the NW estimator when $\kappa(x, x') = K_h(x - x')$ for some smoothing kernel K_h (Figure 2 right).

C Lemmas

Lemma C.1. If $E \in \mathbb{R}^{T \times T}$ is strictly lower triangular,

$$(I_T - E)^{-1} = I_T + E + \cdots + E^{T-1} = \sum_{t=0}^{T-1} E^t \tag{42}$$

Proof. It is sufficient to note

$$(I_T - E) \left(\sum_{t=0}^{T-1} E^t \right) = \sum_{t=0}^{T-1} E^t - \sum_{t=1}^{T} E^t = I_T - E^T = I_T = \left(\sum_{t=0}^{T-1} E^t \right) (I_T - E)$$

where $E^T = 0_{T \times T}$ since E^t has zeros from the t-th subdiagonal. □

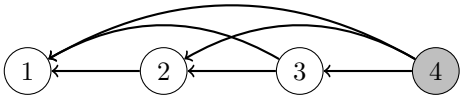
Corollary C.2. Given a DAG of T nodes, let $E \in \mathbb{R}^{T \times T}$ denote the stictly lower triangular adjacency matrix under a topological ordering (i.e., $E_{i,j} \in \mathbb{R}$ is the weight for the directed edge $i \to j$ for $i > j$). Defining the weight of a path as the product of all the involved edge weights (1 for length-0 paths), we have

$$((I_T - E)^{-1})_{i,j} = \text{sum of the weights of all possible paths from } i \text{ to } j \tag{43}$$

Proof. By Lemma C.1, $(I_T - E)^{-1} = I_T + E + \cdots + E^{T-1}$ where each $E^t = I_T E \cdots E$ can be viewed as the usual matrix-form forward algorithm for $t \geq 0$ steps from all T nodes in parallel, computing

$$(E^t)_{i,j} = \text{sum of the weights of all possible paths from } i \text{ to } j \text{ of length } t$$

It is best to see an example. Consider the fully connected DAG over $T = 4$ nodes with unit path weights:



$$E = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The fourth row of E^t computes the t-th step forward weights for all nodes from node 4, e.g., to node 1:

$$\begin{aligned}
E_4^0 &= \begin{bmatrix} \mathbf{0} & 0 & 0 & 1 \end{bmatrix} && \text{(no length-0 path from 4 to 1)} \\
E_4^1 &= \begin{bmatrix} \mathbf{1} & 1 & 1 & 0 \end{bmatrix} && \text{(one length-1 path: } 4 \to 1\text{)} \\
E_4^2 &= \begin{bmatrix} \mathbf{2} & 1 & 0 & 0 \end{bmatrix} && \text{(two length-2 paths: } 4 \to 3 \to 1 \text{ and } 4 \to 2 \to 1\text{)} \\
E_4^3 &= \begin{bmatrix} \mathbf{1} & 0 & 0 & 0 \end{bmatrix} && \text{(one length-3 path: } 4 \to 3 \to 2 \to 1\text{)} \\
\Rightarrow \sum_{t=0}^{3} E_4^t &= \begin{bmatrix} \mathbf{4} & 2 & 1 & 1 \end{bmatrix} && \text{(all-path-sum)}
\end{aligned}$$

□

Corollary C.3. Given $a_1 \dots a_T \in \mathbb{R}^d$, the recursive definition $w_t = a_t + \sum_{i=1}^{t-1} J_{t,i} w_i$ admits the matrix form $W = (I_T - E)^{-1} A$ where $A, W \in \mathbb{R}^{T \times d}$ have the corresponding rows and $E = \text{strictLower}(J)$.

Proof. We can simply write the recursive definition in matrix form and solve the equation:

$$W = A + EW \qquad \Leftrightarrow \qquad A = (I_T - E)W \qquad \Leftrightarrow \qquad W = (I_T - E)^{-1} A$$

A less direct but more insightful proof is to note that the recursive definition builds a fully connected DAG over a_t with edge weights $J_{t,i}$, e.g.,

$$\begin{aligned} w_1 &= a_1 \\ w_2 &= J_{2,1} a_1 + a_2 \\ w_3 &= (J_{3,1} + J_{3,2} J_{2,1}) a_1 + J_{3,2} a_2 + a_3 \end{aligned}$$

Thus $w_t = \sum_{i \le t} R_{t,i} a_i$ where $R_{t,i}$ is the sum of the weights of all possible paths from t to i. By Corollary C.2, $R = (I_T - E)^{-1}$ so the statement follows. □

Lemma C.4 (Product of rank-one updates)**.** Given $(a_1, b_1) \dots (a_T, b_T) \in \mathbb{R}^d \times \mathbb{R}^d$,

$$P_T = \left(I_d - a_1 b_1^\top\right) \cdots \left(I_d - a_T b_T^\top\right) = \prod_{i=1}^{T} \left(I_d - a_i b_i^\top\right) = I_d - W^\top B$$

(left-to-right matmul ordering) where $W = (w_1 \dots w_T) \in \mathbb{R}^{T \times d}$ has rows recursively defined as $w_t = a_t - \sum_{i=1}^{t-1} (a_t^\top b_i) w_i$.

Proof. The statement holds for $T = 1$ since $w_1 = a_1$. For $T \ge 1$

$$P_T = P_{T-1}(I_d - a_T b_T^\top) = P_{T-1} - P_{T-1} a_T b_T^\top = \left(I_d - \sum_{i=1}^{T-1} w_i b_i^\top\right) - w_T b_T^\top = I_d - \sum_{i=1}^{T} w_i b_i^\top$$

where $w_T = P_{T-1} a_T$ follows from the recursive definition:

$$P_{T-1} a_T = \left(I_d - \sum_{i=1}^{T-1} w_i b_i^\top\right) a_T = a_T - \sum_{i=1}^{T-1} (a_T^\top b_i) w_i = w_T$$

□

Corollary C.5. Given $(a_1, b_1) \dots (a_T, b_T) \in \mathbb{R}^d \times \mathbb{R}^d$,

$$\prod_{i=1}^{T} \left(I_d - a_i b_i^\top\right) = I_d - ((I_T + \text{strictLower}(AB^\top))^{-1} A)^\top B$$

Proof. The LHS is $I_d - W^\top B$ by Lemma C.4 where the recursively defined rows $w_t = a_t - \sum_{i=1}^{t-1} (a_t^\top b_i) w_i$ admit the matrix form $W = (I_T + \text{strictLower}(AB^\top))^{-1} A$ by Corollary C.3. □

Lemma C.6. Given $(a_1, b_1, z_1) \dots (a_T, b_T, z_T) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d$ with $N_0 := I_d$,

$$N_T = \left(z_1 b_1^\top\right) \prod_{i=2}^{T} \left(I_d - a_i b_i^\top\right) + \cdots + z_T b_T^\top = \sum_{t=1}^{T} \left(z_t b_t^\top\right) \prod_{i=t+1}^{T} \left(I_d - a_i b_i^\top\right) = U^\top B$$

where $U = (u_1 \dots u_T) \in \mathbb{R}^{T \times d}$ has rows recursively defined as $u_t = z_t - \sum_{i=1}^{t-1} (a_t^\top b_i) u_i$.

Proof. The statement holds for $T = 1$ since $u_1 = z_1$. For $T \ge 1$

$$N_T = N_{T-1}(I_d - a_T b_T^\top) + z_T b_T^\top = N_{T-1} + (z_T - N_{T-1} a_T)\, b_T^\top = \sum_{i=1}^{T-1} u_i b_i^\top + u_T b_T^\top = \sum_{i=1}^{T} u_i b_i^\top$$

where $u_T = z_T - N_{T-1} a_T$ follows from the recursive definition:

$$z_T - N_{T-1} a_T = z_T - \sum_{i=1}^{T-1} (a_T^\top b_i) u_i = u_T$$

□

Lemma C.7 (Rank-1 inhomogeneous recurrence)**.** Given $(a_1, b_1, z_1) \dots (a_T, b_T, z_T) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d$ define

$$S_t = S_{t-1}(I_d - a_t b_t^\top) + z_t b_t^\top$$

for some $S_0 \in \mathbb{R}^{d \times d}$. Then for any segment $1 \le s \le t \le T$ of length $m = t - s + 1$, we can compute S_t from S_{s-1} by a rank $\le m$ update

$$S_t = S_{s-1} + (U_{s:t} - W_{s:t} S_{s-1}^\top)^\top B_{s:t}$$

where $W_{s:t} = (w_s^{(s:t)} \dots w_t^{(s:t)}) \in \mathbb{R}^{m \times d}$ and $U_{s:t} = (u_s^{(s:t)} \dots u_t^{(s:t)}) \in \mathbb{R}^{m \times d}$ have locally recursive rows $w_i^{(s:t)} = a_i - \sum_{j=s}^{i-1} (a_i^\top b_j) w_j^{(s:t)}$ and $u_i^{(s:t)} = z_i - \sum_{j=s}^{i-1} (a_i^\top b_j) u_j^{(s:t)}$ for $i = s \dots t$.

Proof. Denote $M_t = I_d - a_t b_t^\top$ and $X_t = z_t b_t^\top$. Then by unwinding $S_t = S_{t-1} M_t + X_t$, e.g.,

$$\begin{aligned} S_1 &= S_0 M_1 + X_1 \\ S_2 &= S_0 M_1 M_2 + X_1 M_2 + X_2 \\ S_3 &= S_0 M_1 M_2 M_3 + X_1 M_2 M_3 + X_2 M_3 + X_3 \end{aligned}$$

we can verify

$$S_t = S_0 \left(\prod_{i=1}^{t} M_i \right) + \sum_{i=1}^{t} X_i \left(\prod_{j=i+1}^{t} M_j \right) = S_{s-1} \left(\prod_{i=s}^{t} M_i \right) + \sum_{i=s}^{t} X_i \left(\prod_{j=i+1}^{t} M_j \right) \tag{44}$$

By treating $(a_s, b_s, z_s) \dots (a_t, b_t, z_t)$ as an independent length-m sequence, we can write

$$\prod_{i=s}^{t} M_i = \prod_{i=s}^{t} \left(I_d - a_i b_i^\top \right) = I_d - W_{s:t}^\top B_{s:t} \qquad \text{(Lemma C.4)} \tag{45}$$

$$\sum_{i=s}^{t} X_i \left(\prod_{j=i+1}^{t} M_j \right) = \sum_{i=s}^{t} \left(z_i b_i^\top \right) \prod_{j=i+1}^{t} \left(I_d - a_j b_j^\top \right) = U_{s:t}^\top B_{s:t} \qquad \text{(Lemma C.6)} \tag{46}$$

where $W_{s:t}, U_{s:t} \in \mathbb{R}^{m \times d}$ are locally recursive. Plugging (45) and (46) into (44), we have

$$S_t = S_{s-1}(I_d - W_{s:t}^\top B_{s:t}) + U_{s:t}^\top B_{s:t} = S_{s-1} + (U_{s:t} - W_{s:t} S_{s-1}^\top)^\top B_{s:t}$$

□

Corollary C.8. Given $(a_1, b_1, z_1) \dots (a_T, b_T, z_T) \in \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d$ define

$$S_t = S_{t-1}(I_d - a_t b_t^\top) + z_t b_t^\top$$

for some $S_0 \in \mathbb{R}^{d \times d}$. Then for any segment $1 \le s \le t \le T$ of length $m = t - s + 1$, we can compute S_t from S_{s-1} by a rank $\le m$ update

$$\begin{aligned} E_{s:t} &= -\text{strictLower}(A_{s:t} B_{s:t}^\top) \in \mathbb{R}^{m \times m} \\ U_{s:t} &= (I_m - E_{s:t})^{-1} Z_{s:t} \in \mathbb{R}^{m \times d} \\ W_{s:t} &= (I_m - E_{s:t})^{-1} A_{s:t} \in \mathbb{R}^{m \times d} \end{aligned} \qquad S_t = S_{s-1} + (U_{s:t} - W_{s:t} S_{s-1}^\top)^\top B_{s:t} \in \mathbb{R}^{d \times d}$$

Proof. This is just Lemma C.7 using the matrix form of the locally recursive variables $W_{s:t}, U_{s:t} \in \mathbb{R}^{m \times d}$ (Corollary C.3) for reference. □

Lemma C.9 (Monotone DAG closure)**.** Given a DAG of T nodes, let $E \in \mathbb{R}^{T \times T}$ denote the stictly lower triangular adjacency matrix. Let $E_{s:t,s:t} \in \mathbb{R}^{m \times m}$ denote a principal submatrix for some interval $1 \le s \le t \le T$ of length $m = t - s + 1$. Then

$$(I_m - E_{s:t,s:t})^{-1} = ((I_T - E)^{-1})_{s:t,s:t} \tag{47}$$

Proof. For any $i, j \in [s, t]$, Corollary C.2 gives us

$$((I_T - E)^{-1})_{i,j} = \text{sum of the weights of all possible paths from } i \text{ to } j$$

Now, any directed path from i to j in a strictly lower triangular DAG visits only indices between j and i, hence lies entirely in $[s, t]$. Then it is sufficient to use $E_{s:t,s:t}$ instead of the whole E to compute the sum as

$$((I_m - E_{s:t,s:t})^{-1})_{i-s+1,j-s+1} = \text{sum of the weights of all possible paths from } i \text{ to } j$$

The statement (47) follows. □

Lemma C.10 (Pairwise-local adjacency closure)**.** Let $A = (a_1 \dots a_T) \in \mathbb{R}^{T \times d}$ and $B = (b_1 \dots b_T) \in \mathbb{R}^{T \times d}$. Then for any interval $1 \le s \le t \le T$ of length $m = t - s + 1$,

$$(AB^\top)_{s:t,s:t} = A_{s:t} B_{s:t}^\top \in \mathbb{R}^{m \times m} \tag{48}$$

Proof. Directly verify for all $s \le j < i \le t$

$$(AB^\top)_{i,j} = a_i^\top b_j = (A_{s:t})_{i-s+1}^\top (B_{s:t})_{j-s+1} = (A_{s:t} B_{s:t}^\top)_{i-s+1,j-s+1}$$

□

Corollary C.11 (All-path-sum closure)**.** Consider a DAG of T nodes equipped with $A, B \in \mathbb{R}^{T \times d}$ defining adjacency $E = \text{strictLower}(AB^\top) \in \mathbb{R}^{T \times T}$. Let $R = (I_T - E)^{-1} \in \mathbb{R}^{T \times T}$ denote the all-path-sums (Corollary C.2). For any interval $1 \le s \le t \le T$ of length $m = t - s + 1$, define a "local" all-path-sums by $E^{(s:t)} = \text{strictLower}(A_{s:t} B_{s:t}^\top) \in \mathbb{R}^{m \times m}$ and $R^{(s:t)} = (I_m - E^{(s:t)})^{-1} \in \mathbb{R}^{m \times m}$. Then $R^{(s:t)} = R_{s:t,s:t}$.

Proof. For all $s \le j < i \le t$,

$$\begin{aligned}
R_{i,j} &= ((I_T - E)^{-1})_{i,j} \\
&= ((I_m - E_{s:t,s:t})^{-1})_{i-s+1,j-s+1} && \text{(monotone DAG closure, Lemma C.9)} \\
&= ((I_m - E^{(s:t)})^{-1})_{i-s+1,j-s+1} && \text{(pairwise-local adjacency closure, Lemma C.10)} \\
&= R^{(s:t)}_{i-s+1,j-s+1}
\end{aligned}$$

□