# Attention and Position

David I. Inouye

Dynamic Content · Last Updated: October 30, 2026

# Self-attention combines representations from the same sequence

**Arc 6: Sequence-model architectures → Attention and position**

1. Write the complete **self-attention equation**.
2. Unpack its **input-dependent soft lookup**.
3. Extend to **cross-attention**, then restrict access for our next-token task.

Position information comes next.

# Self-attention is one input-dependent matrix operation

Input $X \in \mathbb{R}^{L \times d}$: **one row per position**. Biases omitted.

| Projection | Learned matrix | Result shape |
|---|---|---|
| $Q = XW_Q^\top$ | $W_Q \in \mathbb{R}^{d_k \times d}$ | $L \times d_k$ |
| $K = XW_K^\top$ | $W_K \in \mathbb{R}^{d_k \times d}$ | $L \times d_k$ |
| $V = XW_V^\top$ | $W_V \in \mathbb{R}^{d_v \times d}$ | $L \times d_v$ |

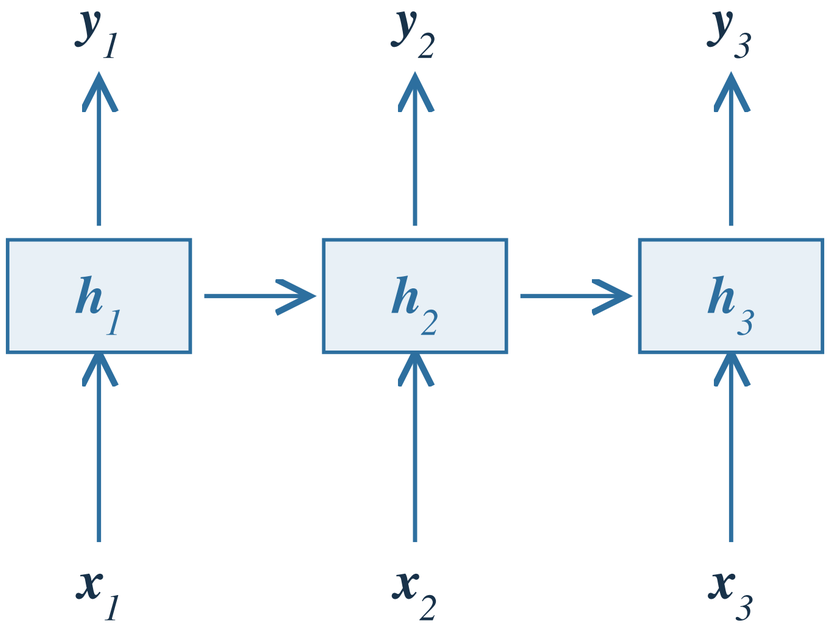
$$A = \mathrm{softmax}_{\mathrm{row}}\left(\frac{QK^\top}{\sqrt{d_k}}\right) \in \mathbb{R}^{L \times L}, \qquad Y = AV \in \mathbb{R}^{L \times d_v}$$

**Attention matrix $A$:** input-dependent weights, one row per output position.

$$\mathrm{SA}(X) = \mathrm{softmax}_{\mathrm{row}}\left(\frac{(XW_Q^\top)(XW_K^\top)^\top}{\sqrt{d_k}}\right)(XW_V^\top)$$

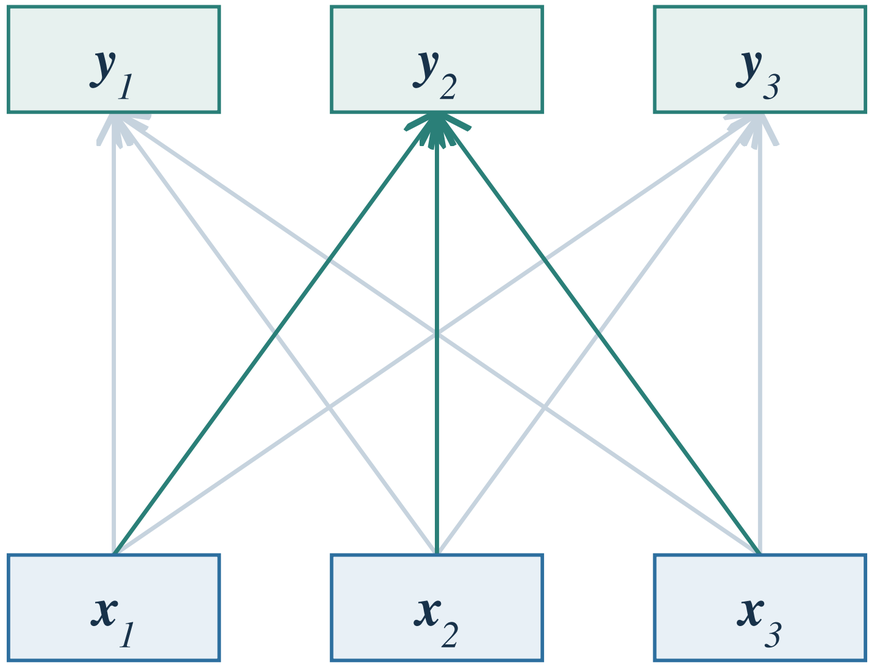
# Self-attention combines inputs directly, with input-dependent weights

### RNN: context follows a state chain




Later states wait for earlier states

### Self-attention: direct input mixing




Each output has input-dependent weights

Each $y_\ell$ combines information from **all rows of** $X$.

The learned projection matrices stay fixed; the combination weights are **input-dependent**.

# Self-attention removes the sequential chain within a layer

| RNN | Self-attention |
|---|---|
| $\boldsymbol{h}_1 \to \boldsymbol{h}_2 \to \cdots \to \boldsymbol{h}_L$ | $X \to \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L\}$ together |
| Distant information travels through successive updates | Each output directly combines all input positions |
| Position $\ell$ waits for the preceding state | Outputs share the same available input collection |

**Shorter paths for learning; parallel operations across positions during training.**

# Column vectors become transposed rows in a sequence matrix

$$X = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_L^\top \end{bmatrix}, \qquad \boldsymbol{q}_\ell = W_Q \boldsymbol{x}_\ell, \qquad Q = XW_Q^\top$$

Likewise $K = XW_K^\top$ and $V = XW_V^\top$. **Example: four tokens, three features per vector.**

$Q \in \mathbb{R}^{4\times 3}$ — Queries

| |
|---|
| $\boldsymbol{q}_1^T = [q_{11},\ q_{12},\ q_{13}]$ |
| $\boldsymbol{q}_2^T = [q_{21},\ q_{22},\ q_{23}]$ |
| $\boldsymbol{q}_3^T = [q_{31},\ q_{32},\ q_{33}]$ |
| $\boldsymbol{q}_4^T = [q_{41},\ q_{42},\ q_{43}]$ |

$K \in \mathbb{R}^{4\times 3}$ — Keys

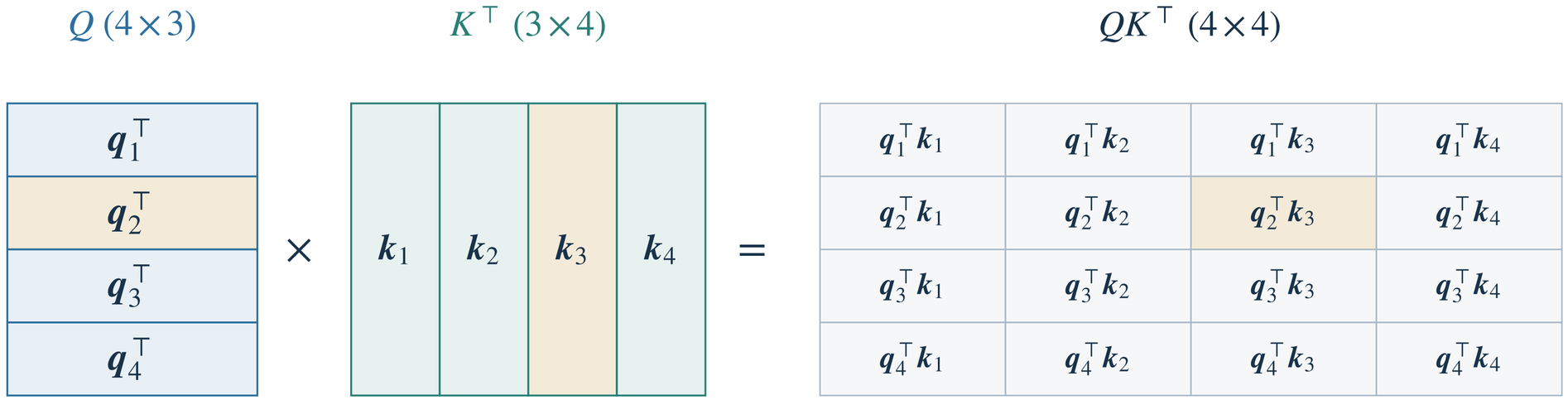
| |
|---|
| $\boldsymbol{k}_1^T = [k_{11},\ k_{12},\ k_{13}]$ |
| $\boldsymbol{k}_2^T = [k_{21},\ k_{22},\ k_{23}]$ |
| $\boldsymbol{k}_3^T = [k_{31},\ k_{32},\ k_{33}]$ |
| $\boldsymbol{k}_4^T = [k_{41},\ k_{42},\ k_{43}]$ |

$V \in \mathbb{R}^{4\times 3}$ — Values

| |
|---|
| $\boldsymbol{v}_1^T = [v_{11},\ v_{12},\ v_{13}]$ |
| $\boldsymbol{v}_2^T = [v_{21},\ v_{22},\ v_{23}]$ |
| $\boldsymbol{v}_3^T = [v_{31},\ v_{32},\ v_{33}]$ |
| $\boldsymbol{v}_4^T = [v_{41},\ v_{42},\ v_{43}]$ |

Four token positions; each row is the transpose of a three-feature column vector.

# Each query–key inner product fills one score-matrix entry




$$(QK^\top)_{23} = \boldsymbol{q}_2^\top \boldsymbol{k}_3 = q_{21}k_{31} + q_{22}k_{32} + q_{23}k_{33}$$

Query 2 matches key 3; the resulting attention weight will multiply value 3.

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad A = \mathrm{softmax}_{\mathrm{row}}(S), \qquad a_{\ell j} = \frac{e^{s_{\ell j}}}{\sum_{r=1}^{L} e^{s_{\ell r}}}$$

**Dot products give scores; row softmax gives weights.**

# Attention builds an input-dependent weighted combination

For each input column vector, form a value: $\boldsymbol{v}_j = f_v(\boldsymbol{x}_j) = W_V \boldsymbol{x}_j$.

$$\boldsymbol{y}_\ell = \sum_{j=1}^{L} a_{\ell j} \boldsymbol{v}_j, \qquad a_{\ell j} \geq 0, \quad \sum_{j=1}^{L} a_{\ell j} = 1$$

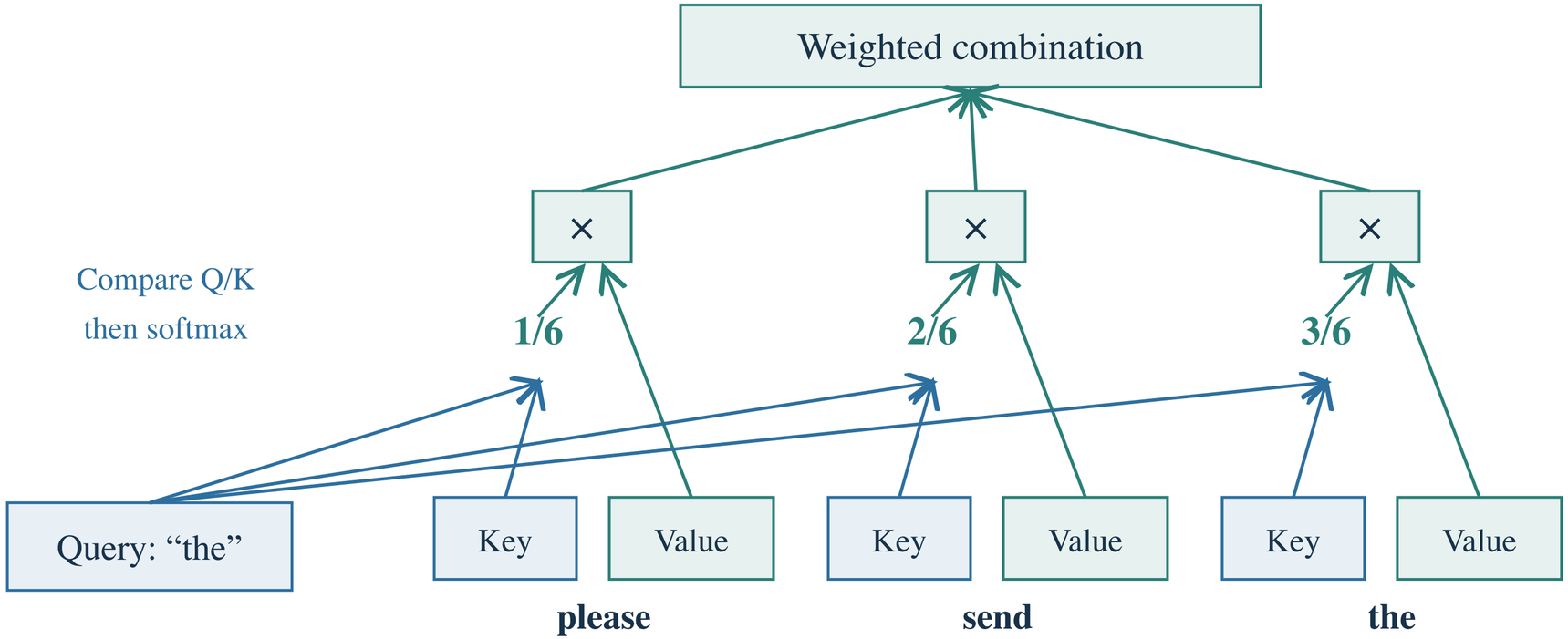
- The score for pair $(\ell, j)$ is a function of **both inputs**: $s_{\ell j} = f_s(\boldsymbol{x}_\ell, \boldsymbol{x}_j)$.
- $a_{\ell j} = \mathrm{softmax}_j(s_{\ell 1}, \ldots, s_{\ell L})$: normalize against **all candidate inputs**.

**How do queries and keys turn these input pairs into useful weights?**

# Attention is a soft lookup in a key–value table

**Dictionary lookup:** an exact key selects one stored value.

**Attention:** a query scores every key, then blends the paired values.




**Query:** features to match. **Key:** matching features. **Value:** features to combine.

# Softmax turns matching scores into relative combination weights

$$s_{\ell j} = \frac{\boldsymbol{q}_\ell^\top \boldsymbol{k}_j}{\sqrt{d_k}}, \qquad a_{\ell j} = \frac{\exp(s_{\ell j} - m_\ell)}{\sum_{k=1}^{L} \exp(s_{\ell k} - m_\ell)}, \qquad m_\ell = \max_k s_{\ell k}$$

- A larger dot product gives a larger weight **relative to the other scores**.
- Subtracting the row maximum preserves the weights and avoids large exponentials.

These weights are **not probabilities that a token is true or important**.

**Next:** calculate one complete weighted combination.

# One weighted combination uses values, not token IDs

Synthetic one-feature query $q = [1]$; the query is at `the`.

| Source token | Key | Score | Softmax weight | Value $v_j$ |
|---|---|---|---|---|
| `please` | $[0]$ | $0$ | $1/6$ | $[1, 0]^\mathsf{T}$ |
| `send` | $[\log 2]$ | $\log 2$ | $2/6$ | $[0, 2]^\mathsf{T}$ |
| `the` | $[\log 3]$ | $\log 3$ | $3/6$ | $[2, 2]^\mathsf{T}$ |

$$y = \tfrac{1}{6}[1, 0]^\mathsf{T} + \tfrac{2}{6}[0, 2]^\mathsf{T} + \tfrac{3}{6}[2, 2]^\mathsf{T} = [7/6, 5/3]^\mathsf{T}$$

**This is a context feature vector.** A later vocabulary head predicts the next token.

# Matrix multiplication computes all output states together

Stack token representations as **rows** of $X \in \mathbb{R}^{L \times d}$.

$$S = QK^\top / \sqrt{d_k}, \qquad A = \mathrm{softmax}_{\mathrm{row}}(S), \qquad Y = AV$$

| Object | Shape | Meaning |
|---|---|---|
| $Q, K$ | $L \times d_k$ | Query/key features per token |
| $QK^\top$ | $L \times L$ | Query rows × key columns |
| $A$ | $L \times L$ | One normalized weight row per query |
| $Y = AV$ | $L \times d_v$ | Combined value features per token |

**One row of $A$ gives the input-dependent weights for one output row of $Y$.**

# Divide by $\sqrt{d_k}$ to keep score fluctuations comparable

Assume independent query/key entries with **mean** $0$ **and variance** $1$.

$$z = \boldsymbol{q}^\top \boldsymbol{k} = \sum_{i=1}^{d_k} q_i k_i, \qquad \mathbb{E}[z] = 0, \qquad \mathrm{Var}(z) = d_k$$

Positive and negative terms partly cancel. **Variances add; standard deviations grow as $\sqrt{d_k}$.**

Dividing by $\sqrt{d_k}$ keeps the score standard deviation at $1$**, for any width**.

| **Two-key scores ($d_k = 20$)** | **Softmax weights** |
|---|---|
| $[0, \sqrt{20}]$ | $[1.1\%, 98.9\%]$ |
| Divide by $\sqrt{20}$: $[0, 1]$ | $[26.9\%, 73.1\%]$ |

**Avoid near-one-hot attention caused merely by width:** saturated softmax has small gradients.

# Cross-attention gets queries and context from different sequences

Query-side inputs $X \in \mathbb{R}^{L_q \times d_x}$; context inputs $C \in \mathbb{R}^{L_c \times d_c}$.

$$Q = XW_Q^\mathsf{T}, \qquad K = CW_K^\mathsf{T}, \qquad V = CW_V^\mathsf{T}$$

$$\mathrm{CA}(X, C) = \mathrm{softmax}_{\mathrm{row}}\left(\frac{(XW_Q^\mathsf{T})(CW_K^\mathsf{T})^\mathsf{T}}{\sqrt{d_k}}\right)(CW_V^\mathsf{T})$$

- Example: **question tokens** query representations of a **passage**.
- Weight matrix: $L_q \times L_c$. Output: one row per query, $L_q \times d_v$.

**Self-attention is the case $C = X$.** The same soft-lookup mechanism applies.

# Next-token prediction determines which context is available

At position $\ell$, we predict $w_{\ell+1}$.

> **Which input tokens may this output attend to?**
>
> **A. Past and current tokens: $w_0, \ldots, w_\ell$**
>
> **B. Later tokens: $w_{\ell+1}, w_{\ell+2}, \ldots$**
>
> **C. Both earlier and later tokens**

**A.** Later inputs include the answer we are trying to predict.

We need an **autoregressive (causal) mask** to enforce that restriction.

# A causal mask removes future positions before softmax

**Key position / input token available to read**

| **Query position** | <BOS> | please | send | the | report |
|---|---|---|---|---|---|
| <BOS> | 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| please | 0 | 0 | $-\infty$ | $-\infty$ | $-\infty$ |
| send | 0 | 0 | 0 | $-\infty$ | $-\infty$ |
| the | 0 | 0 | 0 | 0 | $-\infty$ |
| report | 0 | 0 | 0 | 0 | 0 |

**Add mask to scores BEFORE row-wise softmax**

$$A = \mathrm{softmax}_{\mathrm{row}}(S + M), \qquad Y = AV$$

# Position tells the model where a matching token occurred

**Arc 6: Attention and position**

1. Combine values using query–key matches.
2. Restrict future-token access for next-token prediction.
3. **Locate tokens with position information.**

Then combine information in several learned subspaces.

# Self-attention without positions is permutation equivariant

For any **permutation matrix** $P$, with **no mask or positional encoding**:

$$\mathrm{SA}(PX) = P\,\mathrm{SA}(X)$$

**The outputs are the same, just reordered.**

| Input rows | Output rows |
|---|---|
| Item A · Item B · Item C | $\boldsymbol{y}_A^\mathsf{T} \cdot \boldsymbol{y}_B^\mathsf{T} \cdot \boldsymbol{y}_C^\mathsf{T}$ |
| Item C · Item A · Item B | $\boldsymbol{y}_C^\mathsf{T} \cdot \boldsymbol{y}_A^\mathsf{T} \cdot \boldsymbol{y}_B^\mathsf{T}$ |

**Useful for an unordered set:** changing the storage order should not change what each item means.

# A sequence needs order beyond an unordered collection

| Same words, different order | Requested action |
| --- | --- |
| Send the **report before** the slides. | Report first, slides second. |
| Send the **slides before** the report. | Slides first, report second. |

Focus on **report**: its relationship to **before** has changed.

Without a mask or positions, self-attention gives **report the same output vector in both sentences**—just in a different row.

**We want token representations to reflect order-dependent relationships.**

# Causal masks restrict access; position signals describe location

A fixed causal mask **breaks arbitrary permutation equivariance**: reordering tokens changes which context they may read.

| Mechanism | What it supplies |
|---|---|
| Causal mask | Which earlier/current inputs a query may access |
| Absolute position signal | Where a token occurs in the sequence |
| Relative position signal | How far apart two token positions are |

**The mask does not explicitly encode token indices or pairwise offsets.**

Next: add position features, or rotate query/key features using their positions.

# Additive positions combine token features and location features

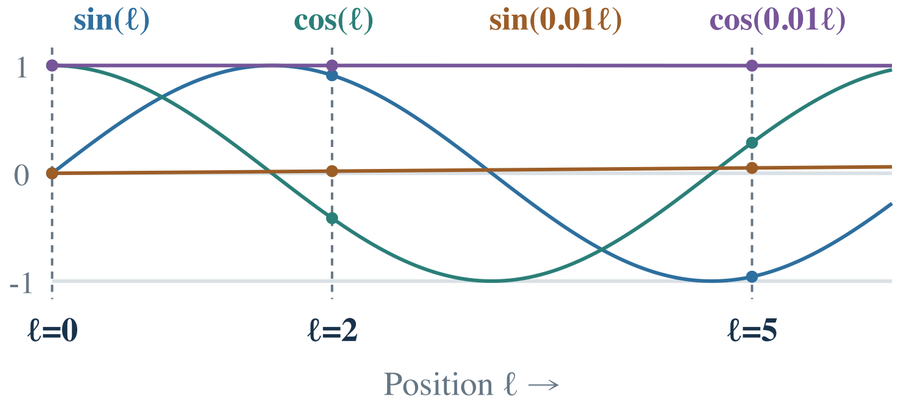
Look up the token's embedding, then **add a vector describing its position**:

$$\boldsymbol{x}_\ell = E[w_\ell, :]^\mathsf{T} + \boldsymbol{p}_\ell, \qquad \boldsymbol{x}_\ell, \boldsymbol{p}_\ell \in \mathbb{R}^d$$

Fixed sinusoids: $p_{\ell,2i} = \sin(\ell\omega_i), p_{\ell,2i+1} = \cos(\ell\omega_i), \omega_i = 10000^{-2i/d}$.

Pair index $i = 0, \ldots, d/2 - 1$; even width $d$.

**d = 4: read four features at each slice**




**Position vector = values at that slice**

| | |
|---|---|
| **ℓ = 0** | [0.00, 1.00, 0.00, 1.00]ᵀ |
| **ℓ = 2** | [0.91, -0.42, 0.02, 1.00]ᵀ |
| **ℓ = 5** | [-0.96, 0.28, 0.05, 1.00]ᵀ |

Order: sine, cosine, sine, cosine

**Same token "the" at positions 0 and 2:** $E[\text{the}, :]^\mathsf{T} + \boldsymbol{p}_0 \neq E[\text{the}, :]^\mathsf{T} + \boldsymbol{p}_2$.

# RoPE rotates query and key pairs by their positions

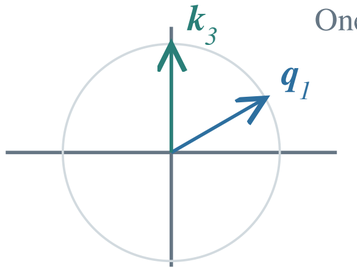
Split each query/key into **2-D feature pairs**; rotate every pair by a position-dependent angle.

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \qquad \tilde{\boldsymbol{q}}_\ell^{(i)} = R(\ell\theta_i)\boldsymbol{q}_\ell^{(i)}$$

$$[\underbrace{q_{\ell,0}, q_{\ell,1}}_{\text{pair 0: } \ell\theta_0}, \quad \underbrace{q_{\ell,2}, q_{\ell,3}}_{\text{pair 1: } \ell\theta_1}, \quad \ldots]^\mathsf{T}, \qquad \theta_i = 10000^{-2i/d_k}$$

**Fast and slow pairs cover different distance scales.** Keys use the same rotations.

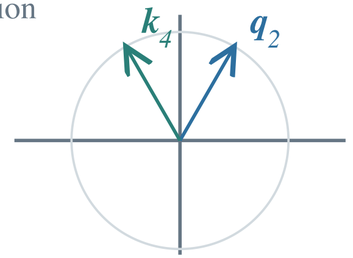
**Positions 1 and 3**




Angle difference 60°; dot product 0.5

One pair: same $[1, 0]^\mathsf{T}$ content; 30° per position

**Positions 2 and 4**




Angle difference 60°; dot product 0.5

# RoPE compares positions through angle differences

Moving both tokens one step rotates both arrows by the same extra angle: **their angle difference stays the same**.

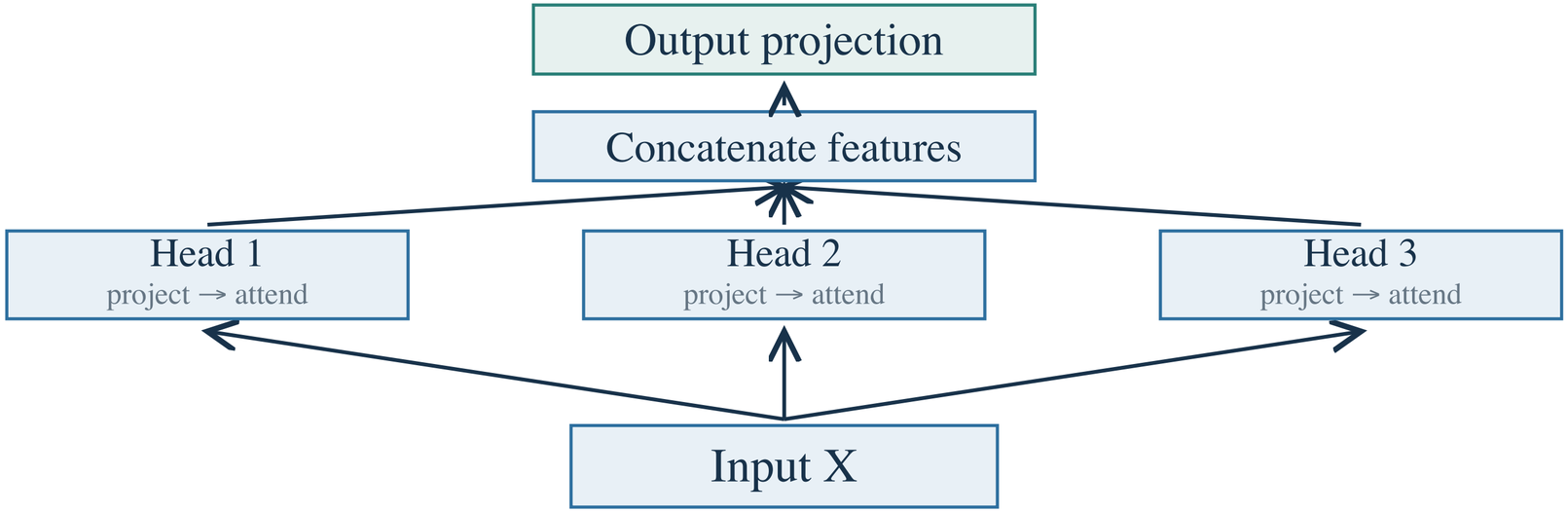
$$s\theta_i - \ell\theta_i = (s-\ell)\theta_i, \qquad (\tilde{\boldsymbol{q}}_\ell^{(i)})^\top \widetilde{\boldsymbol{k}}_s^{(i)} = (\boldsymbol{q}_\ell^{(i)})^\top R((s-\ell)\theta_i)\boldsymbol{k}_s^{(i)}$$

The full dot product **adds the matches from all feature pairs**.

| Why use RoPE? | Practical consequence |
|---|---|
| Relative offset enters the query–key match directly | Same separation gives the same positional comparison, with content held fixed. |
| Fixed rotations, no learned position table | Few extra operations; no separate learned vector for every index. |

**Used in Llama 2:** RoPE is itself a positional encoding, an alternative to adding absolute position vectors.

# Multiple heads learn different ways to combine context




$$Y = \mathrm{Concat}(Y_1, \ldots, Y_b) W_O^\top$$

Each head has its own query/key/value projections; every head obeys the allowed context.

**Next:** combine attention with residual paths, normalization and a token-wise MLP.

# Implementation detail: padding masks exclude absent tokens

Two messages share a batch:

| | Input positions |
|---|---|
| Message A | `please` · `send` · `the` · `report` |
| Message B | `send` · `report` · `<PAD>` · `<PAD>` |

- **Attention key mask:** padded positions cannot supply values to real queries.
- **Loss mask:** padded targets contribute no training loss.

A causal mask still applies within each real prefix.