



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Linear Algebra Foundations for Transformers

Embeddings, projections, self-attention, and next-token prediction

May 3, 2026

Foundations: Vectors and Matrix Views



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Goal and Notational Conventions

These slides summarize the linear algebra needed for embeddings, projections, self-attention, and next-token prediction.

We use **both** conventions:

- **Column-vector convention:** vectors in $\mathbb{R}^n$ written as columns.
- **Row-vector convention:** vectors written as transposes, i.e. rows in $\mathbb{R}^{1 \times n}$.

In transformer practice, both appear:

- mathematics often prefers **column vectors**
- implementations feel more natural in **row form**, where each token is one row

Central teaching point

Left multiplication acts naturally on columns.

Right multiplication acts naturally on rows.

Dot Product, Norm, and Cosine

Let $x, y \in \mathbb{R}^n$.

Dot product

$$\langle x, y \rangle = x^\top y = \sum_{i=1}^n x_i y_i$$

bilinear, symmetric, positive definite: $x^\top x \geq 0$, with equality iff $x = 0$.

Euclidean norm and angle

$$\|x\|_2 = \sqrt{x^\top x}, \quad \cos \theta = \frac{x^\top y}{\|x\|_2 \|y\|_2}$$

Cosine measures **directional similarity** independent of scale.

A Matrix is Both: Columns and Rows

Let $A \in \mathbb{R}^{m \times n}$.

Column view:

$$A = (a_1, \dots, a_n), \quad a_j \in \mathbb{R}^m.$$

Row view:

$$A = \begin{pmatrix} r_1^\top \\ \vdots \\ r_m^\top \end{pmatrix}, \quad r_i \in \mathbb{R}^n.$$

Both views are equivalent — and each makes a different identity obvious.

Matrix Multiplication



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Matrix times Vector: Two Interpretations

Let $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$.

Column combination:

$$Ax = x_1 a_1 + \cdots + x_n a_n.$$

Ax is a linear combination of the columns of A.

Row evaluation:

$$Ax = \begin{pmatrix} r_1^\top x \\ \vdots \\ r_m^\top x \end{pmatrix}.$$

Each entry of Ax is a row-wise dot product with x.

Both statements are simultaneously true.

Row Vector times Matrix

Let $u^\top \in \mathbb{R}^{1 \times m}$ and $A \in \mathbb{R}^{m \times n}$. Then $u^\top A \in \mathbb{R}^{1 \times n}$.

Row combination:

$$u^\top A = u_1 r_1^\top + \cdots + u_m r_m^\top.$$

Column evaluation:

$$u^\top A = (u^\top a_1 \quad \cdots \quad u^\top a_n).$$

Row vector times matrix = linear combination of the rows of A .

Matrix times Matrix

Let $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, so $AB \in \mathbb{R}^{m \times p}$.

Column-wise view:

$$AB = (Ab_1, \dots, Ab_p).$$

Left multiplication by A transforms each *column* of B .

Row-wise view:

$$AB = \begin{pmatrix} r_1^\top B \\ \vdots \\ r_m^\top B \end{pmatrix}.$$

Right multiplication by B transforms each *row* of A .

The key identity for row-wise transformer notation.

Entry-Wise Formula

The (i, j) -entry of AB is

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj} = r_i^\top c_j,$$

where $r_i^\top$ is row i of A and c_j is column j of B .

Core insight

Every entry of a matrix product is a row-column dot product.

Matrix as a Function — Linear Maps

Every matrix $A \in \mathbb{R}^{m \times n}$ defines a function

$$f_A : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad f_A(x) = Ax.$$

Definition (linear map)

$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is **linear** iff for all $x, y \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$,

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y).$$

- f_A is linear (direct check from distributivity).
- Conversely, every linear $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is given by a unique matrix once a basis is fixed: column j of A is $f(e_j)$.

Matrices and linear maps between finite-dim spaces are *the same thing*.

Affine Maps — Linear Plus a Shift

Definition (affine map)

$g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is **affine** iff there exist $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ with

$$g(x) = Ax + b.$$

Equivalently, $g - g(0)$ is linear.

- Linear $\subset$ affine (set $b = 0$).
- Affine maps preserve affine combinations: $g(\sum_i \lambda_i x_i) = \sum_i \lambda_i g(x_i)$ whenever $\sum_i \lambda_i = 1$.
- **Concrete pointer:** the FFN's $h \mapsto hW_1 + b_1$ is affine, not linear — the bias b_1 is what breaks linearity.

Composition Justifies Matrix Multiplication

Let $f_A : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $f_B : \mathbb{R}^m \rightarrow \mathbb{R}^p$ be linear, with matrices A, B . Then

$$(f_B \circ f_A)(x) = B(Ax) = (BA)x.$$

Matrix multiplication is *defined* so that this works:

$$(BA)_{ij} = \sum_k B_{ik} A_{kj}.$$

The formula is not arbitrary — it is forced by the requirement “matrix of composition = product of matrices”.

Closure under composition

- Composition of linear maps is **linear**.
- Composition of affine maps is **affine**: $B(Ax + b_1) + b_2 = (BA)x + (Bb_1 + b_2)$.

Stacking arbitrary numbers of linear (or affine) layers stays inside the same class — which is exactly why nonlinearities are needed.

Matrix Concatenation as a Function

Two ways to assemble matrices into bigger ones:

Horizontal stack. For $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{m \times p}$:

$$[A \mid B] \in \mathbb{R}^{m \times (n+p)}, \quad [A \mid B] \begin{pmatrix} x \\ y \end{pmatrix} = Ax + By.$$

Reads A on the first n coords, B on the last p .

Vertical stack. For $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times n}$:

$$\begin{pmatrix} A \\ B \end{pmatrix} \in \mathbb{R}^{(m+p) \times n}, \quad \begin{pmatrix} A \\ B \end{pmatrix} x = \begin{pmatrix} Ax \\ Bx \end{pmatrix}.$$

Applies A and B in parallel to the same input, stacking outputs.

Forward pointers: vertical stacks underlie MHA's per-head outputs; horizontal stacks appear in fused QKV projections.

Left vs. Right Multiplication

- AX : acts on **columns** » column action
- XB : acts on **rows** » row action

Transpose, Outer Products, Gram Matrices



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Transpose Rules

For compatible matrices,

$$(A^{\top})^{\top} = A, \quad (A + B)^{\top} = A^{\top} + B^{\top}, \quad (AB)^{\top} = B^{\top} A^{\top}.$$

The last identity is especially important.

Example: attention score matrix

$$QK^{\top} = (XW_Q)(XW_K)^{\top} = XW_QW_K^{\top}X^{\top}.$$

Outer Products and Rank-One Matrices

For $u \in \mathbb{R}^m$, $v \in \mathbb{R}^n$, the outer product

$$uv^\top \in \mathbb{R}^{m \times n}, \quad (uv^\top)_{ij} = u_i v_j$$

has rank at most 1.

Outer-product decomposition of a matrix product. If $A = (a_1, \dots, a_n)$ and $B^\top = (b_1, \dots, b_n)$, then

$$AB = \sum_{k=1}^n a_k b_k^\top.$$

Many matrix identities are most naturally read as sums of rank-one pieces.

Gram and Similarity Matrices

Let $X \in \mathbb{R}^{T \times d}$.

Gram matrix of the rows of X :

$$XX^\top \in \mathbb{R}^{T \times T}, \quad (XX^\top)_{ij} = x_i^\top x_j.$$

Gram matrix of the columns of X :

$$X^\top X \in \mathbb{R}^{d \times d}.$$

$XX^\top$ stores *all pairwise row-wise dot products* — the structural blueprint for attention scores.

Bilinear Forms



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

The Adjoint Identity: $\langle Ax, y \rangle = \langle x, A^\top y \rangle$

For $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$:

$$\langle Ax, y \rangle = (Ax)^\top y = x^\top A^\top y = \langle x, A^\top y \rangle,$$

using $(AB)^\top = B^\top A^\top$ on the middle step.

Reading

The transpose moves freely across the inner product: whatever A does on the left of $\langle \cdot, \cdot \rangle$, $A^\top$ does on the right.

This is exactly the move used in the attention score derivation:

$$S_{ij} = (h_i^\top W_Q)(h_j^\top W_K)^\top = \langle W_Q^\top h_i, W_K^\top h_j \rangle.$$

Bilinear Forms — Generalised Inner Products

Definition

$b : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ is **bilinear** iff it is linear in each argument with the other held fixed.

Every bilinear form on $\mathbb{R}^n$ has the form

$$b(x, y) = x^\top M y \quad \text{for a unique } M \in \mathbb{R}^{n \times n}.$$

Inner product is the special case $M = I$.

What's preserved vs. lost

- Linearity in both arguments ✓.
- Symmetry $b(x, y) = b(y, x)$: only iff $M = M^\top$.
- Positive definiteness $b(x, x) \geq 0$: only iff M is PSD.
- “Is $b(x, y)$ large?” — still a meaningful pointwise comparison.



Attention scores are exactly this with $M = W_Q W_K^\top$ — generally non-symmetric, by design (directional attention).

Permutation Matrices



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Permutation Matrices

A **permutation matrix** $P \in \{0, 1\}^{T \times T}$ has exactly one 1 per row and per column.

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \quad P \begin{pmatrix} x_1^\top \\ x_2^\top \\ x_3^\top \end{pmatrix} = \begin{pmatrix} x_2^\top \\ x_3^\top \\ x_1^\top \end{pmatrix}.$$

Two facts we will use repeatedly

- PX permutes the **rows** of X (and $XP^\top$ permutes its columns).
- $P^\top P = I$: permutation matrices are orthogonal.

These show up in the softmax-equivariance lemma (next section) and in the attention-equivariance lemma (later, in positional encoding).



Pointwise Nonlinearities



Pointwise Nonlinearities I — Sigmoid

Linear and affine maps compose to linear / affine. For a stacked network to compute *anything more*, we need **nonlinearities**.

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma : \mathbb{R} \rightarrow (0, 1).$$

Properties:

- Smooth, bounded, monotonically increasing.
- Saturates: $\sigma'(x) \rightarrow 0$ as $|x| \rightarrow \infty$ » vanishing gradients in deep stacks.
- Historical default; mostly displaced by ReLU and its smooth variants.

Applied **pointwise**: $\sigma(z)$ for $z \in \mathbb{R}^n$ means σ applied to each coordinate independently.

Pointwise Nonlinearities II — ReLU and the “LU” Family

All applied **pointwise** to vectors and matrices.

- **ReLU:** $\max(0, x)$. Fast; no positive-side saturation. Risk: *dying ReLU* (neurons stuck at 0).
- **Leaky ReLU:** $\max(\alpha x, x)$ for small $\alpha > 0$. Fixes dying ReLU.
- **ELU / SELU:** smooth negative tail, $\alpha(e^x - 1)$ for $x < 0$.
- **GeLU:** $x \cdot \Phi(x)$, Φ = standard normal CDF. Smooth ReLU. Default in BERT, GPT-2/3, original transformer FFN.
- **SiLU / Swish:** $x \cdot \sigma(x)$. Used in LLaMA et al., usually packaged as *SwiGLU*.

What they share

All are nonlinear scalar functions applied element-wise; they break linearity so that stacked layers can express more than a single affine map.



Softmax

Softmax — Definition and Geometric Picture

For $z \in \mathbb{R}^n$:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}.$$

Maps $\mathbb{R}^n$ to the open simplex $\{p \in \mathbb{R}^n : p_i > 0, \sum_i p_i = 1\}$ — a smooth approximation of $\arg \max$.

Three properties to remember

- **Shift invariance:** $\text{softmax}(z + c\mathbf{1}) = \text{softmax}(z)$ — only differences matter (basis for the “subtract $\max_i z_i$ ” stabilisation trick).
- **Temperature:** $\text{softmax}(z/\tau) \gg$ as $\tau \rightarrow 0$, hard $\arg \max$; as $\tau \rightarrow \infty$, uniform.
- **Equivariant, not invariant:** permuting z permutes $\text{softmax}(z)$ the same way.

Not a pointwise nonlinearity — coordinates are coupled through the denominator.

Row-wise Softmax on Matrices

For $X \in \mathbb{R}^{T \times T}$, define $\text{softmax}(X)$ by applying softmax **row by row**:

$$\text{softmax}(X)_{ij} = \frac{e^{X_{ij}}}{\sum_{k=1}^T e^{X_{ik}}}.$$

Each row of the output is a probability distribution over the T column indices.

Used by attention exactly this way

Each row of the score matrix S becomes one query's distribution over keys.

Row vs. column. The choice of axis matters:

- Row-wise softmax: $\text{softmax}(PX) = P \text{softmax}(X)$ for permutation P (rows are independent).
- A column-wise softmax would be a different operator entirely.

Lemma — Permutation Conjugation Passes Through Softmax

Statement. For every permutation matrix $P \in \{0, 1\}^{T \times T}$ and every $X \in \mathbb{R}^{T \times T}$,

$$\text{softmax}(PXP^\top) = P \text{softmax}(X) P^\top.$$

Proof.

- **Row half:** $\text{softmax}(PX) = P \text{softmax}(X)$ — softmax acts row by row, and P just rearranges which row goes where.
- **Column half:** $\text{softmax}(XP^\top) = \text{softmax}(X) P^\top$ — the row denominator $\sum_k e^{X_{ik}}$ is invariant to column reordering, so each row's softmax just gets its entries relabelled.
- **Compose:** $\text{softmax}(PXP^\top) = P \text{softmax}(XP^\top) = P \text{softmax}(X) P^\top.$ ■

Why this matters

This is the step the attention-equivariance proof glosses over. With this lemma in hand, the chain $A' = \text{softmax}(PSP^\top) = PAP^\top$ is fully rigorous.



Self-Attention



Row-wise vs. Column-wise Token Storage

Row-wise: $X \in \mathbb{R}^{T \times d}$ — each token is one row.

$$X = \begin{pmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{pmatrix}$$

Most natural for implementation-level explanations.

Column-wise: $\tilde{X} \in \mathbb{R}^{d \times T}$ — each token is one column.

$$\tilde{X} = (x_1, \dots, x_T) = X^\top.$$

Mathematically equivalent; they differ only in transpose placement.

Query, Key, Value — Where the Names Come From

The terminology comes from **information retrieval** / dictionary lookup.

Classic lookup: $d[\text{query}]$

A *query* is compared against stored *keys*; on a match, the associated *value* is returned.

Self-attention softens this lookup:

- every key gets a *weight* (not just a single match)
- weights are normalized by softmax
- the output is a *weighted average of values*

In *self-attention*, Q , K , V all come from the *same* X — every token asks: “*which other tokens are keys matching my query?*”

Token $\leftrightarrow$ Row $\leftrightarrow$ Query / Key / Value

Every row of X, Q, K, V is the *same token* through a different learned lens.

row	token	embedding $h_i^\top$	query $h_i^\top W_Q$	key $h_i^\top W_K$	value $h_i^\top W_V$
1	king	$h_1^\top$	$q_1^\top$	$k_1^\top$	$v_1^\top$
2	man	$h_2^\top$	$q_2^\top$	$k_2^\top$	$v_2^\top$
3	woman	$h_3^\top$	$q_3^\top$	$k_3^\top$	$v_3^\top$
4	queen	$h_4^\top$	$q_4^\top$	$k_4^\top$	$v_4^\top$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T	<i>last token</i>	$h_T^\top$	$q_T^\top$	$k_T^\top$	$v_T^\top$

Row i of Q is what token i *asks*; row i of K is what it *advertises*; row i of V is what it *provides*.

Self-Attention: Projections (Row-wise)

Let $X \in \mathbb{R}^{T \times d}$. Choose projection matrices

$$W_Q \in \mathbb{R}^{d \times d_k}, \quad W_K \in \mathbb{R}^{d \times d_k}, \quad W_V \in \mathbb{R}^{d \times d_v}.$$

Define

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V.$$

Right multiplication transforms every token *row* into a query / key / value row. So $Q, K \in \mathbb{R}^{T \times d_k}$ and $V \in \mathbb{R}^{T \times d_v}$.

Score Entry as an Inner Product

Rows of the projected matrices are

$$\text{row } i \text{ of } Q = h_i^\top W_Q, \quad \text{row } j \text{ of } K = h_j^\top W_K.$$

The decomposition

$$S_{ij} = q_i^\top k_j = (h_i^\top W_Q)(h_j^\top W_K)^\top = h_i^\top W_Q W_K^\top h_j = \langle W_Q^\top h_i, W_K^\top h_j \rangle.$$

A **scalar product** between the *learned query vector* $W_Q^\top h_i$ and the *learned key vector* $W_K^\top h_j$, both in $\mathbb{R}^{d_k}$.

(The middle step uses the transpose rule $(AB)^\top = B^\top A^\top$.)

Geometric Reading of the Score

In the query/key subspace $\mathbb{R}^{d_k}$,

$$q_i^\top k_j = \|q_i\|_2 \|k_j\|_2 \cos \theta_{ij}.$$

Two geometric readings

1. **Projection.** $q_i^\top k_j$ is the length of the projection of q_i onto the direction of k_j , scaled by $\|k_j\|_2$.
2. **Magnitude $\times$ alignment.** Large iff *both* vectors are long *and* they point similarly.

“Token i attends to token j ” means: in the learned Q/K subspace, the query of i has a large component in the direction of the key of j .

The Full Score Matrix (screenshot view)

$$Q = \begin{pmatrix} h_1^\top W_Q \\ \vdots \\ h_T^\top W_Q \end{pmatrix}, \quad K^\top = (W_K^\top h_1 \quad \cdots \quad W_K^\top h_T).$$

$$QK^\top = \begin{pmatrix} h_1^\top W_Q W_K^\top h_1 & \cdots & h_1^\top W_Q W_K^\top h_T \\ \vdots & \ddots & \vdots \\ h_T^\top W_Q W_K^\top h_1 & \cdots & h_T^\top W_Q W_K^\top h_T \end{pmatrix}.$$

Every entry is an inner product in the query/key subspace. The score matrix $S = QK^\top$ depends *entirely on the input X* itself.

Scores as a Learned Bilinear Form

Set $M = W_Q W_K^\top \in \mathbb{R}^{d \times d}$. Then

$$S_{ij} = h_i^\top M h_j.$$

The score matrix is a *learned bilinear form* on the embedding space.

- **Asymmetry.** In general $M \neq M^\top$, so $S_{ij} \neq S_{ji}$ — attention is *directional*. (“The cat chased the mouse”: the relation between *cat* and *mouse* is not symmetric.)
- **Low rank.** $\text{rank}(M) \leq d_k$. Factoring M into two $d \times d_k$ pieces is an explicit rank bottleneck.
- **Why two matrices, not one.** W_Q and W_K can learn *different* embeddings for “asking” and “being asked about” — the asymmetry is a *feature*, not a cost.

Self-Attention: Scores and Output

Score matrix.

$$S = QK^{\top} \in \mathbb{R}^{T \times T}, \quad S_{ij} = q_i^{\top} k_j.$$

All pairwise query-key dot products.

Scaled dot-product attention.

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

Softmax is applied **row-wise**; each row of A sums to 1.

Output.

$$\text{Attention}(X) = AV \in \mathbb{R}^{T \times d_v}.$$

Row i : $(AV)_{i,\cdot} = \sum_{j=1}^T A_{ij} v_j^{\top}$ — a weighted average of value rows.

Why the Factor $1/\sqrt{d_k}$?

Variance argument

If query and key entries have variance of order 1, then $q_i^\top k_j$ has variance of order d_k .

Scaling by $\sqrt{d_k}$:

- keeps logits in a numerically stable range,
- prevents softmax from becoming too sharp too early.

The Same Attention in Column-wise Notation

If tokens are columns, $\tilde{X} \in \mathbb{R}^{d \times T}$, projections are applied on the **left**:

$$\tilde{Q} = W_Q \tilde{X}, \quad \tilde{K} = W_K \tilde{X}, \quad \tilde{V} = W_V \tilde{X}.$$

The score matrix becomes

$$\tilde{Q}^\top \tilde{K} \in \mathbb{R}^{T \times T}.$$

The two notations differ by transpose placement only; the underlying mathematics is identical.

Multi-Head Attention



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

From One Head to Many — Multi-Head Attention

Multi-head attention (MHA) is **not a fundamentally new mechanism**. It is the simple attention from §4 run in parallel on *disjoint slices of the embedding space*, with a final mixing projection that recomposes the slices back into a d -dimensional output.

Three structural changes from single-head

1. **Slice the embedding space** into H chunks of size $d_k = d/H$.
2. **Per-head Q/K/V projections**, each of shape $d \times d_k$.
3. **New output projection** $W_O \in \mathbb{R}^{d \times d}$ that mixes head outputs.

Concretely (GPT-2 small): $d = 768$, $H = 12$, $d_k = 64$.

Slicing and Per-Head Projections

Every head $h = 1, \dots, H$ gets its own *narrower* projection matrices:

$$W_Q^{(\ell,h)}, W_K^{(\ell,h)}, W_V^{(\ell,h)} \in \mathbb{R}^{d \times d_k}.$$

Q/K/V parameter budget is unchanged

$$3 \cdot H \cdot d \cdot d_k = 3 \cdot H \cdot d \cdot (d/H) = 3d^2.$$

Multi-head is **not** “more expensive per projection” — it is the *same* parameter budget rearranged into H parallel subspaces.

The scaling denominator changes from $\sqrt{d_k}$ already used per head — each head now works in a d_k -dimensional subspace, not the full d .

The Output Projection W_O

After running attention per head and concatenating the H per-head outputs back to dimension d , one extra matrix mixes them:

$$W_O^{(\ell)} \in \mathbb{R}^{d \times d}.$$

The only “new” piece

W_O has **no analogue in single-head attention**. It is the reason multi-head attention has $4d^2$ parameters where single-head had $3d^2$: the extra d^2 is exactly W_O .

Parameter Accounting: Single-Head vs. Multi-Head

Variant	Q / K / V	W_O	Total per block
Single-head	$3d^2$	—	$3d^2$
Multi-head, H heads, $d_k = d/H$	$3H \cdot d \cdot d_k = 3d^2$	d^2	$4d^2$

Q/K/V parameter count is identical between the two; multi-head adds exactly the W_O mixing matrix on top.

Toy example

$d = 6$, $H = 2$, $d_k = 3$: $4 \cdot 36 = 144$ MHA parameters per block. Independent of T .

Why This Reshaping Matters

Each head produces its own $T \times T$ routing matrix $A^{(h)}$.

Specialisation across heads

Training encourages different heads to discover different relationships: coreference, syntax, position, bracket matching, ...

- Single-head attention cannot represent multiple simultaneous routing patterns.
- Multi-head gets it *essentially for free* in Q/K/V compute, at the cost of the one extra W_O projection.

The forward pass is the natural generalisation of the single-head version: the same five steps (project, score, scale, softmax, weight), now done H times in parallel, with concat + projection at the end.

MHA Forward Pass: Per-Head Projections

Let $X \in \mathbb{R}^{T \times d}$ be the input to block ℓ . For each head $h = 1, \dots, H$:

$$Q^{(h)} = X W_Q^{(\ell, h)}, \quad K^{(h)} = X W_K^{(\ell, h)}, \quad V^{(h)} = X W_V^{(\ell, h)}.$$

Shape check.

$$\underbrace{(T \times d)}_X \cdot \underbrace{(d \times d_k)}_W \rightarrow (T \times d_k).$$

So $Q^{(h)}, K^{(h)}, V^{(h)} \in \mathbb{R}^{T \times d_k}$. ✓

MHA Forward Pass: Scores, Softmax, Weighted Values

Attention scores (per head):

$$S^{(h)} = \frac{Q^{(h)} (K^{(h)})^\top}{\sqrt{d_k}} \in \mathbb{R}^{T \times T}.$$

Softmax (row-wise; each row of $A^{(h)}$ sums to 1):

$$A^{(h)} = \text{softmax}(S^{(h)}) \in \mathbb{R}^{T \times T}.$$

Weighted values:

$$Z^{(h)} = A^{(h)} V^{(h)} \in \mathbb{R}^{T \times d_k}.$$

Same five steps as single-head, run H times in parallel.

$T \times T$ is an Activation, Not a Weight

$S^{(h)}$ and $A^{(h)}$ have shape $T \times T$ — but they are **activations**, not weights.

- Computed fresh for every input; never stored as parameters.
- No learnable matrix in the whole transformer has a $T \times T$ shape.
- The T^2 in “transformers are quadratic in sequence length” is about *compute and memory* during the forward pass, not about parameter count.

Concatenate Heads and Project

Concatenate the H per-head outputs along the feature dimension:

$$Z = [Z^{(1)} \mid Z^{(2)} \mid \dots \mid Z^{(H)}] \in \mathbb{R}^{T \times H d_k} = \mathbb{R}^{T \times d}.$$

Mix the slices with the output projection:

$$U = Z W_O^{(\ell)} \in \mathbb{R}^{T \times d}.$$

Shape check: $(T \times d) \cdot (d \times d) \rightarrow (T \times d)$. ✓

W_O is what allows information discovered in one head's subspace to influence other coordinates of the output — the slices are no longer disjoint after this step.

MHA Parameter Count per Block

- H heads $\times$ 3 projection matrices $\times (d \times d_k)$ entries each:

$$3H \cdot d \cdot d_k = 3H \cdot d \cdot (d/H) = 3d^2.$$

- Plus $W_O \in \mathbb{R}^{d \times d}$, contributing d^2 .

Total: $4d^2$ per block

Independent of sequence length T .

Toy ($d = 6$, $H = 2$, $d_k = 3$): $4 \cdot 36 = 144$ MHA parameters per block.



Feed-Forward Network



The Feed-Forward Block — Where Knowledge Lives

After multi-head attention, each token has absorbed information from other tokens. Attention has done its job of *gathering* — but it has not yet done much *thinking*. All attention does is produce weighted sums of existing vectors. Turning “a pronoun that has absorbed information about its antecedent” into “the concept of the animal it refers to, enriched with facts about animals” — that is the job of the FFN.

Attention vs. FFN, in one breath

Attention gathers information across positions.

FFN processes it, per token, in isolation.

Most of what we call “the model’s knowledge” lives in FFN weights.

FFN: Parameters and the Expansion Factor

A two-layer MLP applied independently to each token. For each block ℓ :

$$W_1^{(\ell)} \in \mathbb{R}^{d \times 4d}, \quad b_1^{(\ell)} \in \mathbb{R}^{4d},$$

$$W_2^{(\ell)} \in \mathbb{R}^{4d \times d}, \quad b_2^{(\ell)} \in \mathbb{R}^d.$$

The expansion factor

The first matrix expands the dimension by a factor of 4 (sometimes 8 in newer architectures): blow the token up to $4 \times$ its size, apply a non-linearity, compress back down.

GPT-3 ($d = 12,288$): hidden dim inside the FFN is $4d = 49,152$.

FFN: Forward Pass

Applied **per row**. For each token $h_t \in \mathbb{R}^d$:

$$\text{FFN}(h_t) = \varphi(h_t \cdot W_1 + b_1) \cdot W_2 + b_2 \in \mathbb{R}^d,$$

where φ is a pointwise non-linearity (ReLU, GeLU, SwiGLU — the choice does not affect shapes).

On the whole matrix (broadcasting b_1, b_2 across rows):

$$\text{FFN}(H) = \varphi(H \cdot W_1 + b_1) \cdot W_2 + b_2 \in \mathbb{R}^{T \times d}.$$

Shape check.

- $H \cdot W_1: (T \times d) \cdot (d \times 4d) \rightarrow (T \times 4d)$
- add b_1 , apply φ : same shape
- $\dots \cdot W_2: (T \times 4d) \cdot (4d \times d) \rightarrow (T \times d) \checkmark$

Position-Wise: No Mixing Across Tokens

W_1, W_2, b_1, b_2 are the *same* matrices applied to every row.

The crucial observation

The FFN is a **position-wise** map: it treats each of the T tokens independently, in parallel. Nothing in the FFN routes information across token positions.

This is the mathematical reason attention has to exist as a separate mechanism — it is the *only* sub-layer that mixes rows with each other.

Attention » cross-token routing

FFN » per-token computation

FFN as Key–Value Memory

A useful interpretability framing of the FFN: a per-token **key–value memory**.

- W_1 scores the token against $4d$ internal “keys”.
- φ gates which keys are active — a soft mask saying “these match”.
- W_2 produces the output as a weighted combination of $4d$ stored “values”.

Routing vs. storage

Attention is routing — it needs relatively few parameters.

FFN is storage — it needs a lot.

Where the Parameters Live: Attention vs. FFN

Sub-layer	Params per block	Toy ($d = 6$)
MHA ($Q/K/V + W_O$)	$4d^2$	144
FFN (W_1, W_2, b_1, b_2)	$8d^2 + 5d$	318

FFN holds roughly two-thirds of the per-block budget

For $d = 768$: MHA ≈ 2.4 M, FFN ≈ 4.7 M parameters per block.

Over GPT-3's 96 layers: *tens of billions* of parameters dedicated to per-token transformation.

Most of the model's knowledge lives in FFN weights, not in attention.

The Full Transformer Block



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Putting the Block Together (Pre-LN + Residuals)

Every block applies layer-normalisation, then a sub-layer, then a residual:

$$H' = H_{\ell-1} + \text{MHA}(\text{LN}_1(H_{\ell-1})),$$

$$H_{\ell} = H' + \text{FFN}(\text{LN}_2(H')).$$

Shape preservation

Every intermediate quantity is in $\mathbb{R}^{T \times d}$. Shape is preserved end-to-end — blocks can be stacked freely.

Across the depth of the model: attention » FFN » attention » FFN » ...
gather, then think, then gather, then think.

Total Parameters per Block: $\approx 12d^2$

Sum across the sub-layers:

$$\underbrace{4d^2}_{\text{MHA}} + \underbrace{4d}_{\text{two LNs}} + \underbrace{8d^2 + 5d}_{\text{FFN}} = 12d^2 + 9d \approx 12d^2.$$

Independent of sequence length T

Toy ($d = 6$): $12 \cdot 36 + 9 \cdot 6 = 432 + 54 = 486$ parameters per block.

Roughly $\frac{1}{3}$ attention, $\frac{2}{3}$ FFN. Stack L blocks for a total of $\approx 12Ld^2$ transformer-stack parameters.

Positional Encoding — Motivation



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

The Blind Spot — Attention Sees a Bag of Words

Take the sentence the cat sat on the mat and scramble the word order.

What changes in attention?

Nothing structurally. The set of pairwise scores $q_i^\top k_j$ is *identical*, just relabelled. Each score depends on **which two tokens** are involved — never on **where they sit**.

- Attention's input is, mathematically, an *unordered set* of token vectors.
- Without help, the transformer is just a very expensive bag-of-words model.

We will now make this precise as a lemma — and then fix it.

Lemma — Self-Attention is Permutation Equivariant

Recap. Self-attention is the function

$$\text{Attention} : \mathbb{R}^{T \times d} \longrightarrow \mathbb{R}^{T \times d_v}$$

parametrised by fixed (learned) matrices $W_Q, W_K \in \mathbb{R}^{d \times d_k}$ and $W_V \in \mathbb{R}^{d \times d_v}$, defined by

$$\text{Attention}(X) = \text{softmax}\left(\frac{(XW_Q)(XW_K)^\top}{\sqrt{d_k}}\right) XW_V,$$

with softmax applied **row-wise**: $\text{softmax}(M)_{ij} = \exp(M_{ij}) / \sum_k \exp(M_{ik})$.

Lemma (permutation equivariance)

For every permutation matrix $P \in \{0, 1\}^{T \times T}$ and every $X \in \mathbb{R}^{T \times d}$,

$$\text{Attention}(PX) = P \cdot \text{Attention}(X) \in \mathbb{R}^{T \times d_v}.$$

Proof of the Lemma

Write $Q = XW_Q$, $K = XW_K$, $V = XW_V$, $S = QK^\top / \sqrt{d_k}$, $A = \text{softmax}(S)$, so $\text{Attention}(X) = AV$.

Step 1 (projections). Matrix multiplication is associative, hence

$$Q' = (PX)W_Q = P(XW_Q) = PQ \in \mathbb{R}^{T \times d_k},$$

and analogously $K' = PK \in \mathbb{R}^{T \times d_k}$, $V' = PV \in \mathbb{R}^{T \times d_v}$.

Step 2 (scores). Using $(AB)^\top = B^\top A^\top$,

$$S' = \frac{Q'(K')^\top}{\sqrt{d_k}} = \frac{(PQ)(PK)^\top}{\sqrt{d_k}} = P \cdot \frac{QK^\top}{\sqrt{d_k}} \cdot P^\top = P S P^\top \in \mathbb{R}^{T \times T}.$$

Step 3 (softmax). Right-multiplication by $P^\top$ permutes the columns of S , but the row-sum normaliser in softmax does not depend on column order. Hence $\text{softmax}(S P^\top) = \text{softmax}(S) P^\top$. Left-multiplication by P permutes rows and passes through softmax trivially. Together:

$$A' = \text{softmax}(P S P^\top) = P \text{softmax}(S) P^\top = P A P^\top \in \mathbb{R}^{T \times T}.$$

Step 4 (output). Using $P^\top P = I$ (P is orthogonal),

$$\text{Attention}(PX) = A'V' = (P A P^\top)(P V) = P A (P^\top P) V = P A V = P \cdot \text{Attention}(X). \blacksquare$$

What Equivariance Means for Position

Each output row is a function only of *which tokens are present*, never of their order.

Token `cat`'s enriched representation

Identical content whether the sequence is
the cat sat, sat the cat, or cat sat the
— only its *row index* in the output matrix changes.

- Each token's output depends only on the **multiset** of other tokens.
- LayerNorm and FFN are position-wise, so the *whole block* is permutation equivariant.
- Stacking blocks does not fix this — equivariance composes.

Consequence: with no extra signal, the model literally cannot tell
“*John loves Mary*” from “*Mary loves John*”.

The Fix — Inject Position Into the Tokens

We don't want to change attention itself — equivariance is what gives us parallelism, length generalisation, and no recurrence.

The trick

Change the **input**, so that the same token at different positions becomes a different vector:

$$X \rightsquigarrow X + E_{\text{pos}}, \quad E_{\text{pos}} \in \mathbb{R}^{T \times d}.$$

Now in general

$$\text{Attention}(P(X + E_{\text{pos}})) \neq P \cdot \text{Attention}(X + E_{\text{pos}}),$$

because $PE_{\text{pos}} \neq E_{\text{pos}}$. The equivariance is **broken on purpose**.

How do we choose E_{pos} ? — next section.

Positional Encoding — Methods



Three Standard Approaches

Let $X \in \mathbb{R}^{T \times d}$ be the token-embedding matrix.

Method	Where it acts	Params	Used by
Sinusoidal	$H_0 = X + P_{\text{sin}}$ (computed)	0	“Attention Is All You Need”
Learned absolute	$H_0 = X + E_{\text{pos}}[0 : T]$	$n_{\text{max}} \cdot d$	GPT-2, GPT-3
RoPE	rotate Q, K <i>inside</i> attention	0	LLaMA, most modern frontier models

The framing matters

Two of these modify the **input** ($H_0 = X + P$); RoPE does *not* — it leaves $H_0 = X$ untouched and injects position later, inside the attention computation. There is no single “ P matrix” that covers all three.

(3a) Sinusoidal Encoding (Vaswani et al., 2017)

The position matrix is **computed**, not learned. For position $t = 0, \dots, T - 1$ and frequency index $k = 0, \dots, d/2 - 1$:

$$P[t, 2k] = \sin\left(\frac{t}{10000^{2k/d}}\right), \quad P[t, 2k + 1] = \cos\left(\frac{t}{10000^{2k/d}}\right).$$

Then $H_0 = X + P \in \mathbb{R}^{T \times d}$.

Three nice properties

- **Zero parameters.** P is a fixed mathematical object.
- **Any length.** For a new T , re-evaluate the formula — nothing stored to resize.
- **Relative positions emerge.** $P[t + \Delta, :]$ is a linear function of $P[t, :]$ (a fixed rotation depending only on Δ), so the model can learn relative-position rules from absolute encodings.

(3b) Learned Absolute (GPT-2 / GPT-3 Style)

A second lookup table, keyed on **position** instead of token identity:

$$E_{\text{pos}} \in \mathbb{R}^{n_{\text{max}} \times d}.$$

For input length $T \leq n_{\text{max}}$:

$$P = E_{\text{pos}}[0 : T, :] \in \mathbb{R}^{T \times d}, \quad H_0 = X + P.$$

The one n_{max} in the entire architecture

This is the *only* weight whose shape depends on the maximum sequence length.

- Parameter count: $n_{\text{max}} \cdot d$.
- Shorter input: prefix of rows; the rest get zero gradient on that pass.
- Longer input: **no row to look up** — the model literally cannot process it.

This is the hard context-window ceiling of pre-RoPE models like GPT-2 and GPT-3.

(3c) RoPE — Rotary Position Embeddings (Su et al., 2021)

Don't add anything to X . Instead, **rotate** Q and K **inside attention** by a position-dependent angle.

For each consecutive 2-coordinate slice $(2k, 2k + 1)$ of row q_t (and k_t), apply the 2D rotation

$$R(t \cdot \omega_k) = \begin{pmatrix} \cos(t\omega_k) & -\sin(t\omega_k) \\ \sin(t\omega_k) & \cos(t\omega_k) \end{pmatrix}, \quad \omega_k = \frac{1}{10000^{2k/d}}.$$

Why it became the modern default

- **Zero learned parameters**, no $n_{\max}$ -shaped matrix.
- The score $q_t^\top k_s$ depends only on the *relative* offset $t - s$ — relative encoding, not absolute.
- Extrapolates beyond the training length (with degrading quality; mitigated by NTK / YaRN scaling).

• Used by LLaMA, Mistral, Qwen, and most current frontier models.

Tying It Back to the Equivariance Lemma

Recap of the lemma: pure self-attention satisfies

$$\text{Attention}(PX) = P \cdot \text{Attention}(X).$$

Position information has to come from *breaking* this symmetry somewhere — and the three methods break it at qualitatively different places.

Input-side break (sinusoidal, learned absolute)

Leave Attention alone; change the input $X \rightsquigarrow X + P_{\text{pos}}$. The lemma still holds for the function Attention — we just stop feeding it permutation-symmetric input, because $P \cdot P_{\text{pos}} \neq P_{\text{pos}}$ in general.

Attention-side break (RoPE)

Modify Attention itself. Rotating Q and K by position-dependent angles makes the score $\tilde{S}_{ts} = q_t^\top R(s-t) k_s$ explicitly position-dependent. The lemma no longer applies — it described the *unmodified* function.

State of the Art Today (2024–2026)

RoPE has won for autoregressive LLMs

Used by essentially every modern open-weights model: LLaMA 1/2/3, Mistral / Mixtral, Qwen 2/3, DeepSeek V2/V3, Yi, Gemma, Falcon, PaLM 2.

GPT-4 family unconfirmed but widely believed to use RoPE or a close variant.

The other methods, briefly:

- **Learned absolute** (GPT-2, GPT-3, original BERT) — legacy. The hard $n_{\max}$ ceiling rules it out for long-context models.
- **Sinusoidal** — did not survive the scaling era; no advantage over RoPE.
- **ALiBi** (BLOOM, MPT-7B) — a different relative scheme; lost mindshare.

Why RoPE won: zero parameters, relative positioning, length extrapolation (with NTK / YaRN scaling), and clean composition into the existing attention dot product.

Caveat: at the long-context frontier ($> 32k$ tokens) even RoPE needs scaling tricks — the race is not fully over.



Similarity, Orthogonality, Projections



Dot Product vs. Cosine Similarity

$$x^\top y = \|x\|_2 \|y\|_2 \cos \theta.$$

The dot product depends on both **magnitude** and **direction**.

Cosine similarity removes magnitude:

$$\text{cosim}(x, y) = \frac{x^\top y}{\|x\|_2 \|y\|_2}.$$

Scale invariance. For $\alpha, \beta > 0$:

$$\text{cosim}(\alpha x, \beta y) = \text{cosim}(x, y).$$

Euclidean distance is *not* scale invariant.

Cosine Similarity: Geometric Picture

With unit-norm versions $\hat{x} = x/\|x\|_2$ and $\hat{y} = y/\|y\|_2$,

$$\text{cosim}(x, y) = \hat{x}^\top \hat{y}.$$

Cosine similarity is just the dot product after ℓ_2 -normalization.

On the unit sphere ($\|x\|_2 = \|y\|_2 = 1$):

$$\|x - y\|_2^2 = 2 - 2 \text{cosim}(x, y).$$

For normalized vectors, Euclidean distance and cosine similarity give *equivalent rankings*.

Why Dot Product, Not Cosine?

Self-attention uses $q_i^\top k_j$, *not* cosine similarity — and the reason is not just computational cheapness.

Cosine throws away a learned degree of freedom

W_Q and W_K learn *both* the **direction** and **magnitude** of q_i and k_j . Forcing unit norm discards that.

Norms can meaningfully encode:

- $\|q_i\|$ — how strongly token i *wants* to attend somewhere
- $\|k_j\|$ — how loudly token j *advertises* itself

Numerical stability of the raw dot product is handled by softmax plus the $1/\sqrt{d_k}$ scaling — cosine normalization is not needed for that.

Why Not Euclidean Distance?

Distance would also seem natural, and

$$\|q_i - k_j\|_2^2 = \|q_i\|_2^2 + \|k_j\|_2^2 - 2 q_i^\top k_j.$$

So distance reduces to the dot product plus two self-norms. Why not use it?

- **Composition.** $q_i^\top k_j = h_i^\top W_Q W_K^\top h_j$ is a clean bilinear function of the learned weights; a squared distance loses that form.
- **Softmax compatibility.** Softmax expects *larger = more relevant*. Distance is larger for *less* similar pairs, and its offset $\|q_i\|^2 + \|k_j\|^2$ varies per token.
- **High-dimensional behavior.** When d_k is large, the self-norm terms tend to dominate and drown out the pairwise term.

Why Not General ℓ_r -Norms?

$\|x - y\|_r$ for $r \neq 2$ is a legitimate distance — but it answers a different question.

- $\|x - y\|_2$: how far apart are the points?
- $\cos \theta$: how aligned are the directions?

Why $r = 2$ is special:

$$\|x\|_2^2 = x^\top x.$$

Euclidean geometry comes from an inner product, which gives access to angles, orthogonality, projections, and Gram matrices. General ℓ_r -norms do not interact as naturally with matrix multiplication.

Orthogonality and Projections

Orthogonality. $x \perp y$ iff $x^\top y = 0$ — no component in each other's direction.

Scalar projection of x onto u :

$$\frac{x^\top u}{\|u\|_2}.$$

Vector projection:

$$\text{proj}_u(x) = \frac{x^\top u}{u^\top u} u.$$

Dot products measure how much one vector points along another — a large query-key dot product means strong alignment.

Summary and Takeaways



Hochschule Aalen
UNIVERSITY OF APPLIED SCIENCES

Summary: Core Identities

Matrix times vector

$$Ax = \sum_{j=1}^n x_j a_j = \begin{pmatrix} r_1^\top x \\ \vdots \\ r_m^\top x \end{pmatrix}$$

Row vector times matrix

$$u^\top A = \sum_{i=1}^m u_i r_i^\top$$

Matrix times matrix

$$AB = (Ab_1, \dots, Ab_p) = \begin{pmatrix} r_1^\top B \\ \vdots \\ r_m^\top B \end{pmatrix}$$

Gram matrix: $(XX^\top)_{ij} = x_i^\top x_j$.

Summary: Self-Attention Pipeline

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$S = QK^\top$$

$$\text{Attention}(X) = \text{softmax}\left(\frac{S}{\sqrt{d_k}}\right) V$$

In one breath

Right multiplication » row action » Q/K/V projections » Gram-style $QK^\top$ » softmax + value averaging.

Pitfalls to Watch For

- confusing rows and columns
- forgetting dimension compatibility
- thinking AB and BA are usually the same
- forgetting that $(AB)^\top = B^\top A^\top$, not $A^\top B^\top$
- confusing dot-product similarity with Euclidean distance
- assuming cosine similarity and Euclidean distance are equivalent **without** normalization
- thinking attention scores are cosine similarities — they are scaled dot products

One-Line Takeaways

- Left multiplication transforms **columns**.
- Right multiplication transforms **rows**.
- QK^T stores all query-key dot products.
- Cosine compares direction; dot product compares direction *and* magnitude.
- Self-attention is linear algebra plus softmax.