Transformer ML

What is a Transformer?

A Transformer is a neural network architecture built around **attention** rather than recurrence or convolution
It processes a *set* of token embeddings in parallel and learns which tokens should pay attention to which others to solve a task (translate, summarize, classify, generate, etc.).
Canonical reference: *Vaswani et al., 2017 — "Attention Is All You Need"*.

Why it works (the intuition)

- **Content-based routing of information.** Instead of fixed local windows (CNNs) or step-by-step state passing (RNNs), attention lets any token directly "query" the entire sequence for relevant context.
- **Parallelism.** All positions are processed simultaneously, making training fast and hardware-friendly.
- **Long-range dependency handling.** Tokens many steps apart can interact in one hop via attention.

Core components

1. **Token & positional embeddings**

   Tokens x_i get embeddings $e_i \in \mathbb{R}^d$. Because attention is permutation-invariant, we add a position signal (sinusoidal or learned; many modern models use rotary embeddings/ALiBi).

2. **Self-Attention (single head)**

   From each input $x \in \mathbb{R}^d$, make:

   $$Q = xW_Q, \quad K = xW_K, \quad V = xW_V \ \ (W_{*} \in \mathbb{R}^{d \times d_k})$$

   For a sequence matrix $X \in \mathbb{R}^{n \times d}$,

   $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V$$

   where M is a mask (e.g., causal mask for autoregressive decoding).

3. **Multi-head attention**

   Run h attention "heads" with separate projections; concatenate and mix:

   $$\text{MHA}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O$$

4. **Position-wise feedforward**

   Two-layer MLP applied per position: $\mathrm{FFN}(x) = \sigma(xW_1 + b_1)W_2 + b_2$ (often with GELU).

5. **Residual connections & LayerNorm**

   Stabilize and speed training:

   $X \leftarrow X + \mathrm{MHA}(X)$, then LayerNorm; repeat with FFN.

6. **Stacks & variants**

   - **Encoder–decoder** (original Transformer): encoder produces context; decoder attends to encoder and itself (causally).
   - **Decoder-only** (e.g., GPT): for next-token prediction and general text generation.
   - **Encoder-only** (e.g., BERT): for bidirectional understanding tasks.

Training objectives (typical)

- **Autoregressive (language modeling):** maximize $\sum_t \log p(x_t \mid x_{<t})$.
- **Masked language modeling (BERT):** predict masked tokens from full context.
- **Sequence-to-sequence (T5):** text-to-text tasks with a span-corruption objective.

Computational profile (rationale & limits)

- **Complexity:** self-attention is $O(n^2)$ in time and memory with sequence length n. This is the main bottleneck for very long inputs.
- **Mitigations:** efficient attention kernels (e.g., FlashAttention), sparse/linear attention, chunking, recurrence, or hierarchical schemes; plus architectural tweaks (RoPE, ALiBi) to extrapolate length.

Minimal mental model (pseudocode)

```
for layer in layers:
  X = X + MultiHeadSelfAttention(LayerNorm(X))   # (causal mask for decoders)
  X = X + FeedForward(LayerNorm(X))
logits = Linear(X)  # next-token scores (decoder) or task head (encoder)
```

Example

0) Tokens, embeddings, and positions

Sequence (length $n = 3$): $t_1 =$"Buy", $t_2 =$"milk", $t_3 =$"and".

Model dim $d = 2$ (keep it tiny). Intuition: first coord ≈ "foodness", second ≈ "action/connector".

Token embeddings

- $e(\text{Buy}) = [0.1,\ 1.0]$ (mostly action)
- $e(\text{milk}) = [1.0,\ 0.1]$ (strong foodness)
- $e(\text{and}) = [0.6,\ 0.6]$ (connector-ish)

Positional encodings (small learned shifts)

- $p_1 = [0.0,\ 0.0]$
- $p_2 = [+0.1,\ -0.1]$
- $p_3 = [-0.1,\ +0.1]$

Inputs to attention $X = E + P$ (rows are tokens):

$$x_1 = [0.1, 1.0], \quad x_2 = [1.1, 0.0], \quad x_3 = [0.5, 0.7].$$

1) Causal self-attention

Use simple projections:

$$W_Q = I, \quad W_K = I, \quad W_V = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

So $Q = X,\ K = X,\ V = XW_V$ (doubling the "foodness" in values).

Values $V = [v_1; v_2; v_3]$ with rule $[a, b] \mapsto [2a, b]$:

- $v_1 = [0.2, 1.0]$, $v_2 = [2.2, 0.0]$, $v_3 = [1.0, 0.7]$.

Scaled scores $S = \frac{QK^\top}{\sqrt{2}}$. First compute dot products $XX^\top$:

$$XX^\top = \begin{bmatrix} 1.01 & 0.11 & 0.75 \\ 0.11 & 1.21 & 0.55 \\ 0.75 & 0.55 & 0.74 \end{bmatrix} \Rightarrow S \approx \begin{bmatrix} 0.7149 & 0.0778 & 0.5303 \\ 0.0778 & 0.8550 & 0.3890 \\ 0.5303 & 0.3890 & 0.5230 \end{bmatrix}.$$

Causal mask (no peeking ahead):

- Row1 → $[0.7149, -\infty, -\infty]$
- Row2 → $[0.0778, 0.8550, -\infty]$
- Row3 → $[0.5303, 0.3890, 0.5230]$

Row-wise softmax → attention A

(Using $e^{0.7149} \approx 2.044, e^{0.0778} \approx 1.081, e^{0.8550} \approx 2.351, e^{0.5303} \approx 1.699, e^{0.3890} \approx 1.475, e^{0.5230} \approx 1.687$)

- Row1: $[1, 0, 0]$
- Row2: $[0.3149,\ 0.6851,\ 0]$
- Row3: $[0.3497,\ 0.3034,\ 0.3470]$

Context $Y = AV$ (weighted sums of v_j):

- $y_1 = v_1 = [0.2,\ 1.0]$
- $y_2 = 0.3149\, v_1 + 0.6851\, v_2 \approx [1.5702,\ 0.3149]$
- $y_3 = 0.3497\, v_1 + 0.3034\, v_2 + 0.3470\, v_3 \approx [1.0844,\ 0.5926]$

Interpretation: at "**and**" (row 3), attention mixes **Buy**, **milk**, and itself. Because W_V amplifies the food dimension, the context vector leans food-ward—nudging the model to continue the **grocery list** with another food.

2) Residual + LayerNorm (pre-norm)

Add X and Y, then LN per token (use $\gamma=1, \beta=0$ for clarity).

Residual

$$R = X + Y = \begin{bmatrix} 0.3 & 2.0 \\ 2.6702 & 0.3149 \\ 1.5844 & 1.2926 \end{bmatrix}$$

LayerNorm each row to zero-mean, unit-variance:

- Row1 $[0.3, 2.0] \to [-1, \ +1]$
- Row2 $[2.6702, 0.3149] \to [+1, \ -1]$
- Row3 $[1.5844, 1.2926] \to [+1, \ -1]$

Call this Z:

$$Z = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix}.$$

3) Position-wise Feed-Forward (FFN)

Tiny ReLU MLP:

$$W_1 = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}.$$

Compute per row (you can verify by hand):

- For $[-1, 1]$: ReLU$([-1, 1, 0]) = [0, 1, 0]$ → FFN output $[0, 1]$.
- For $[1, -1]$: ReLU$([1, -1, -2]) = [1, 0, 0]$ → FFN output $[1, 0.5]$.

Thus

$$\mathrm{FFN}(Z) = \begin{bmatrix} 0 & 1 \\ 1 & 0.5 \\ 1 & 0.5 \end{bmatrix}.$$

Residual + LN again

$$U = Z + \mathrm{FFN}(Z) = \begin{bmatrix} -1 & 2 \\ 2 & -0.5 \\ 2 & -0.5 \end{bmatrix} \xrightarrow{\text{LN}} H = \begin{bmatrix} -1 & +1 \\ +1 & -1 \\ +1 & -1 \end{bmatrix}.$$

4) Output head → logits for the next token

Let the (toy) vocab be $\{\text{eggs}, \text{bread}, \text{now}\}$ with

$$W_{\text{out}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \end{bmatrix}.$$

We care about **position 3** (after "and") because it conditions the **next word**.

With $h_3 = [1, -1]$:

- Logits: $L_3 = h_3 W_{\text{out}} = [1, -1, -2]$.

Largest logit is the **first** → the model predicts "**eggs**" as the next token.

Why this matches real life

1. **Content routing:** At “and”, attention looks back mainly to “milk” (a food).
2. **Value projection W_V:** doubles the “foodness”, making the context vector point toward other groceries.
3. **FFN & head:** map that direction to a concrete token—**eggs**—a common item co-listed with milk.

You can change the sentence (e.g., “Pack laptop and …”), swap embeddings to make “techness” high for “laptop”, and the same math would push the next token toward “charger”. The steps—Q/K/V, causal mask, softmax weights, value mix, residual/LN, FFN, residual/LN, output—remain identical; only the numbers (and thus the prediction) change.