Transformers I: Self-Attention, Positional Encoding and Transformer Blocks

Replacing recurrence with attention between all tokens

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

Why Transformers?

Where we are

Last lectures built sequence models on **recurrence** — RNNs, LSTMs, and seq2seq with **attention**:

Where we are

Last lectures built sequence models on **recurrence** — RNNs, LSTMs, and seq2seq with **attention**:

$h_t = f(h_{t-1}, x_t)$ (encoder: a state carried step by step)

Where we are

Last lectures built sequence models on **recurrence** — RNNs, LSTMs, and seq2seq with **attention**:

$$h_t = f(h_{t-1}, x_t)$$ (encoder: a state carried step by step)

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t$$ (decoder attends over encoder states h_t)

Where we are

Last lectures built sequence models on **recurrence** — RNNs, LSTMs, and seq2seq with **attention**:

$$h_t = f(h_{t-1}, x_t) \qquad \text{(encoder: a state carried step by step)}$$

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t \qquad \text{(decoder attends over encoder states } h_t\text{)}$$

Attention already lets the decoder look **directly** at any source position. This lecture asks: what if attention were the **whole** model — and we dropped recurrence entirely?

The problem with recurrence

D DERIVATION

Attention fixed the **bottleneck**, but the encoder states h_t are still built **sequentially**:

The problem with recurrence

D DERIVATION

Attention fixed the **bottleneck**, but the encoder states h_t are still built **sequentially**:

$h_1 \to h_2 \to h_3 \to \ldots \to h_T$ (each h_t waits for h_{t-1})

The problem with recurrence

D DERIVATION

Attention fixed the **bottleneck**, but the encoder states h_t are still built **sequentially**:

$$h_1 \to h_2 \to h_3 \to \ldots \to h_T \qquad \text{(each } h_t \text{ waits for } h_{t-1}\text{)}$$

$h_t = f(h_{t-1}, x_t)$ is **inherently serial**: you cannot compute step t before step $t-1$. On modern parallel hardware (GPUs) this wastes the machine — training time grows with sequence length T.

The problem with recurrence

D DERIVATION

Attention fixed the **bottleneck**, but the encoder states h_t are still built **sequentially**:

$$h_1 \to h_2 \to h_3 \to \ldots \to h_T \qquad \text{(each } h_t \text{ waits for } h_{t-1}\text{)}$$

$h_t = f(h_{t-1}, x_t)$ is **inherently serial**: you cannot compute step t before step $t-1$. On modern parallel hardware (GPUs) this wastes the machine — training time grows with sequence length T.

we want the representation of every token computed in parallel

The Transformer idea

D DERIVATION

Replace the recurrent update with a function of **all** tokens at once:

The Transformer idea

D DERIVATION

Replace the recurrent update with a function of **all** tokens at once:

$$\underbrace{h_t = f(h_{t-1}, x_t)}_{\text{RNN: depends on the previous state}} \longrightarrow \underbrace{z_i = f(x_1, ..., x_T)}_{\text{Transformer: attends to all tokens}}$$

The Transformer idea

D DERIVATION

Replace the recurrent update with a function of **all** tokens at once:

$$\underbrace{h_t = f(h_{t-1}, x_t)}_{\text{RNN: depends on the previous state}} \longrightarrow \underbrace{z_i = f(x_1, ..., x_T)}_{\text{Transformer: attends to all tokens}}$$

Each token builds its new representation by **directly attending** to every other token:

The Transformer idea

D DERIVATION

Replace the recurrent update with a function of **all** tokens at once:

$$\underbrace{h_t = f(h_{t-1}, x_t)}_{\text{RNN: depends on the previous state}} \longrightarrow \underbrace{z_i = f(x_1, ..., x_T)}_{\text{Transformer: attends to all tokens}}$$

Each token builds its new representation by **directly attending** to every other token:

Self-attention relates positions **within the same sequence**. No state to carry — every z_i is computed independently and in parallel (Vaswani et al. 2017, “Attention Is All You Need”).

Three ways to mix information

V VISUAL

	RNN	CNN	Transformer
context mechanism	hidden state h_t	receptive field	attention
reach per layer	all past (decayed)	fixed kernel k	all positions
parallel over positions?	no — serial	yes	yes
content-dependent mixing?	partly	no — fixed	yes

Three ways to mix information

V VISUAL

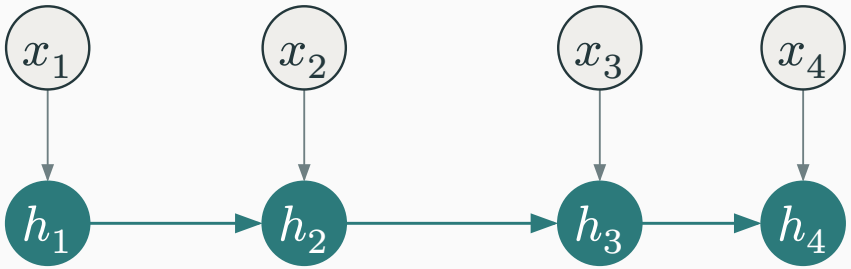
	RNN	CNN	Transformer
context mechanism	hidden state h_t	receptive field	attention
reach per layer	all past (decayed)	fixed kernel k	all positions
parallel over positions?	no — serial	yes	yes
content-dependent mixing?	partly	no — fixed	yes

the Transformer keeps the CNN's parallelism **and** gains content-dependent, full-range mixing

Recurrence vs attention, side by side

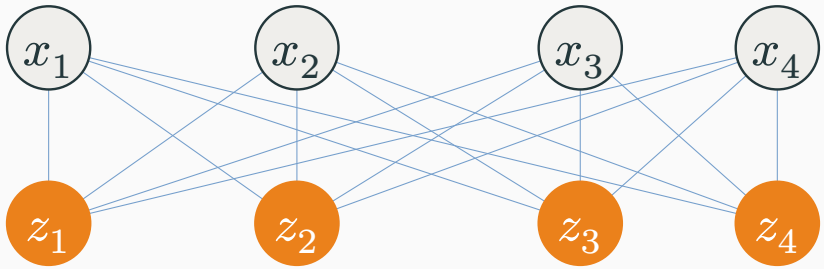
V VISUAL

path $i \to j$: $O(|i-j|)$ steps



RNN — sequential

path $i \to j$: $O(1)$ – one layer

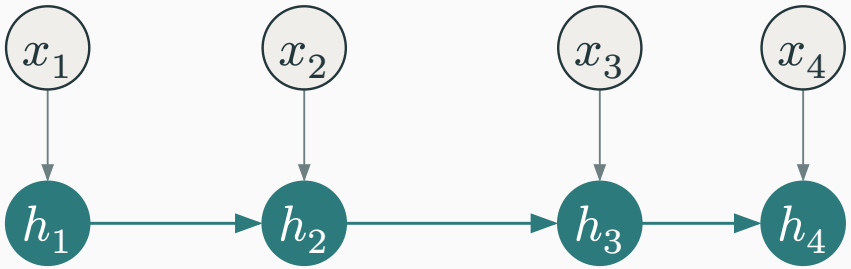


self-attention — all-to-all

Recurrence vs attention, side by side

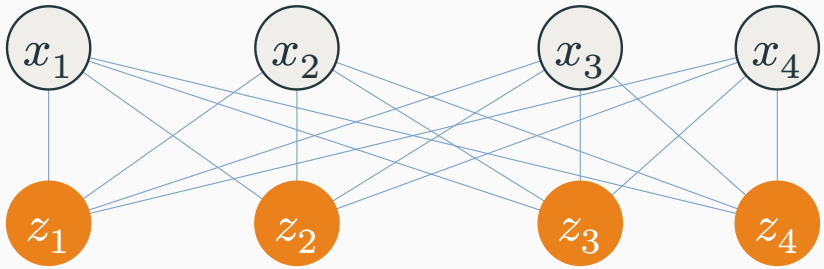
V VISUAL

path $i \to j$: $O(|i-j|)$ steps



RNN — sequential

path $i \to j$: $O(1)$ – one layer



self-attention — all-to-all

the RNN routes information along a **chain**; self-attention wires **every token to every token** in one layer

Attention as a soft lookup

Recall: query, keys, values

D DERIVATION

Attention is a **differentiable dictionary lookup**. Given a query q and T key–value pairs:

Recall: query, keys, values

D DERIVATION

Attention is a **differentiable dictionary lookup**. Given a query q and T key–value pairs:

$s_i = q^\top k_i$ (score: how well the query matches key i)

Recall: query, keys, values

D DERIVATION

Attention is a **differentiable dictionary lookup**. Given a query q and T key–value pairs:

$$s_i = q^\top k_i \qquad \text{(score: how well the query matches key } i\text{)}$$

$$\alpha = \mathrm{softmax}(s) \qquad o = \sum_{i=1}^{T} \alpha_i \, v_i$$

Recall: query, keys, values

D DERIVATION

Attention is a **differentiable dictionary lookup**. Given a query q and T key–value pairs:

$$s_i = q^\top k_i \qquad \text{(score: how well the query matches key } i\text{)}$$

$$\alpha = \mathrm{softmax}(s) \qquad o = \sum_{i=1}^{T} \alpha_i \, v_i$$

a **soft** lookup: instead of picking one entry, return a weighted blend of **all** values

The Q / K / V intuition

D DERIVATION

Three learned roles, one per vector:

The Q / K / V intuition

D DERIVATION

Three learned roles, one per vector:

Vector	Question it answers	Role
query q	what am I looking for?	probe
key k_i	what do I contain?	address
value v_i	what do I pass on?	content

The Q / K / V intuition

D DERIVATION

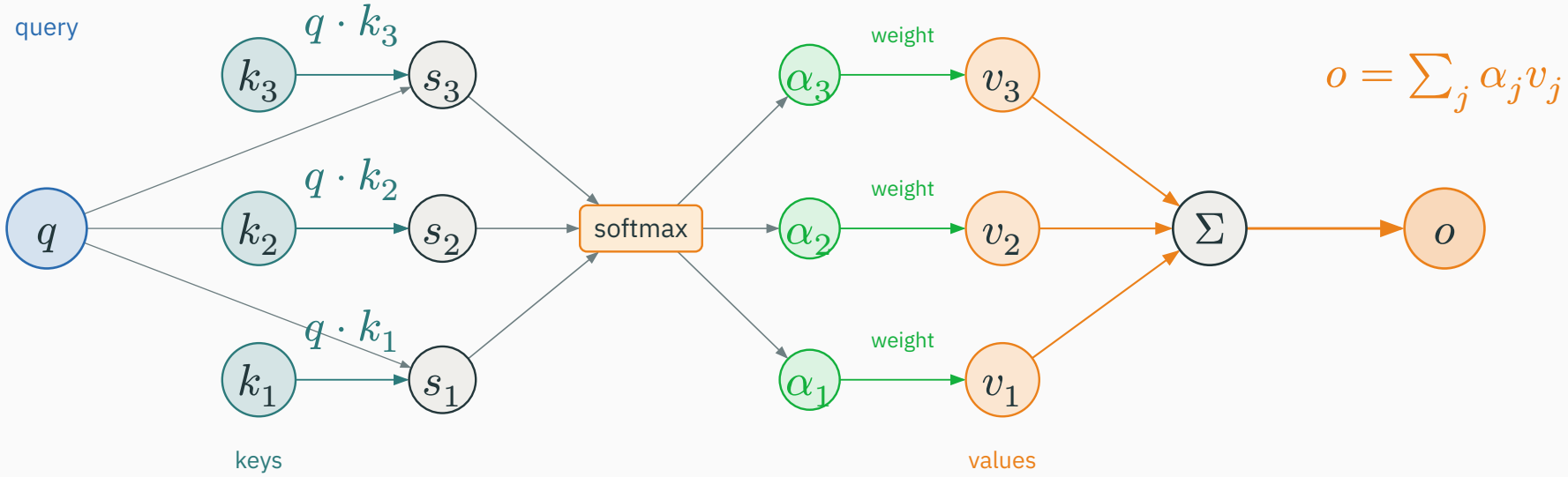
Three learned roles, one per vector:

Vector	Question it answers	Role
query q	what am I looking for?	probe
key k_i	what do I contain?	address
value v_i	what do I pass on?	content

match q against every k_i to decide **how much** of each v_i to retrieve — all differentiable

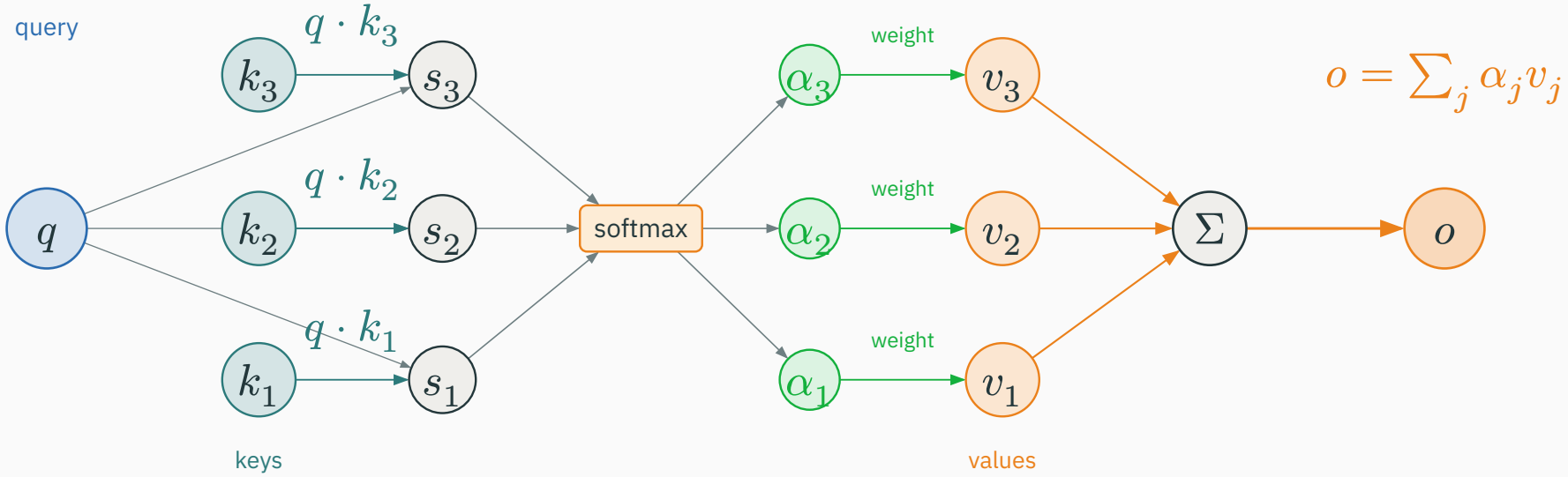
Attention as a soft dictionary

V VISUAL



Attention as a soft dictionary

V VISUAL



compare query to keys $\longrightarrow$ scores $\longrightarrow$ softmax weights α $\longrightarrow$ blend the values into o

Worked attention: the setup

D DERIVATION

$q = \begin{pmatrix} 1 & 0 \end{pmatrix}$, three keys, three values:

Worked attention: the setup

D DERIVATION

$q = (1 \;\; 0)$, three keys, three values:

$$k_1 = (1 \;\; 0), \quad k_2 = (0 \;\; 1), \quad k_3 = (1 \;\; 1)$$

Worked attention: the setup

D DERIVATION

$q = (1 \;\; 0)$, three keys, three values:

$$k_1 = (1 \;\; 0), \quad k_2 = (0 \;\; 1), \quad k_3 = (1 \;\; 1)$$

$$v_1 = (10 \;\; 0), \quad v_2 = (0 \;\; 10), \quad v_3 = (10 \;\; 10)$$

Worked attention: the setup

D DERIVATION

$q = (1 \;\; 0)$, three keys, three values:

$$k_1 = (1 \;\; 0), \quad k_2 = (0 \;\; 1), \quad k_3 = (1 \;\; 1)$$

$$v_1 = (10 \;\; 0), \quad v_2 = (0 \;\; 10), \quad v_3 = (10 \;\; 10)$$

Scores are dot products of the query with each key:

$$s = (q^\top k_1, q^\top k_2, q^\top k_3) = (1, 0, 1)$$

Worked attention: softmax and output

D DERIVATION

Softmax the scores $s = (1, 0, 1)$:

Worked attention: softmax and output

D DERIVATION

Softmax the scores $s = (1, 0, 1)$:

$$\alpha = \mathrm{softmax}(1, 0, 1) = \frac{e^1, e^0, e^1}{e^1 + e^0 + e^1} \approx (0.422, 0.155, 0.422)$$

Worked attention: softmax and output

D DERIVATION

Softmax the scores $s = (1, 0, 1)$:

$$\alpha = \mathrm{softmax}(1, 0, 1) = \frac{e^1, e^0, e^1}{e^1 + e^0 + e^1} \approx (0.422, 0.155, 0.422)$$

Blend the values with these weights:

$$o = 0.422\, v_1 + 0.155\, v_2 + 0.422\, v_3$$

Worked attention: softmax and output

D DERIVATION

Softmax the scores $s = (1, 0, 1)$:

$$\alpha = \mathrm{softmax}(1, 0, 1) = \frac{e^1, e^0, e^1}{e^1 + e^0 + e^1} \approx (0.422, 0.155, 0.422)$$

Blend the values with these weights:

$$o = 0.422\, v_1 + 0.155\, v_2 + 0.422\, v_3$$

$$o \approx (8.45, 5.78)$$

Worked attention: softmax and output

D DERIVATION

Softmax the scores $s = (1, 0, 1)$:

$$\alpha = \mathrm{softmax}(1, 0, 1) = \frac{e^1, e^0, e^1}{e^1 + e^0 + e^1} \approx (0.422, 0.155, 0.422)$$

Blend the values with these weights:

$$o = 0.422\, v_1 + 0.155\, v_2 + 0.422\, v_3$$

$$o \approx (8.45, 5.78)$$

keys 1 and 3 match the query, so o leans toward v_1 and v_3 — a soft blend, not a hard pick

Scaled dot-product attention

D DERIVATION

Stack the queries, keys and values into matrices and do it all at once:

Scaled dot-product attention

D DERIVATION

Stack the queries, keys and values into matrices and do it all at once:

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

Scaled dot-product attention

D DERIVATION

Stack the queries, keys and values into matrices and do it all at once:

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

this single expression is the central operation of the Transformer

Scaled dot-product attention

D DERIVATION

Stack the queries, keys and values into matrices and do it all at once:

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

this single expression is the central operation of the Transformer

$QK^\top$ is every query dotted with every key; softmax normalizes each row; the result weights V

Why divide by $\sqrt{d_k}$?

D DERIVATION

Suppose query and key entries are independent with mean 0, variance 1. Then a single score

Why divide by $\sqrt{d_k}$?

D DERIVATION

Suppose query and key entries are independent with mean 0, variance 1. Then a single score

$$q^\top k = \sum_{i=1}^{d_k} q_i\, k_i \qquad \Rightarrow \qquad \mathrm{Var}(q^\top k) = d_k$$

Why divide by $\sqrt{d_k}$?

D DERIVATION

Suppose query and key entries are independent with mean 0, variance 1. Then a single score

$$q^\top k = \sum_{i=1}^{d_k} q_i\, k_i \qquad \Rightarrow \qquad \mathrm{Var}(q^\top k) = d_k$$

Large $d_k \to$ scores of magnitude $\sim \sqrt{d_k} \to$ softmax saturates into a near one-hot spike $\to$ gradients through it vanish. Learning stalls.

Why divide by $\sqrt{d_k}$?

D DERIVATION

Suppose query and key entries are independent with mean 0, variance 1. Then a single score

$$q^\top k = \sum_{i=1}^{d_k} q_i\, k_i \qquad \Rightarrow \qquad \mathrm{Var}(q^\top k) = d_k$$

Large $d_k \to$ scores of magnitude $\sim \sqrt{d_k} \to$ softmax saturates into a near one-hot spike $\to$ gradients through it vanish. Learning stalls.

divide by $\sqrt{d_k}$ to hold the score variance at ≈ 1, whatever d_k is

Matrix shapes

V VISUAL

One self-attention layer, sequence length T, model width d, head width d_k:

Matrix shapes

V VISUAL

One self-attention layer, sequence length T, model width d, head width d_k:

Tensor	Shape	How
input X	$T \times d$	one row per token
W_Q, W_K, W_V	$d \times d_k$	learned projections
Q, K, V	$T \times d_k$	$Q = XW_Q$, etc.
scores $A = \mathrm{softmax}\left(QK^\top / \sqrt{d_k}\right)$	$T \times T$	token-to-token weights
output $O = AV$	$T \times d_k$	attended representations

Interactive: attention lookup

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Drag the query and watch the scores $q^\top k_i$, the softmax weights α_i, and the blended output $o = \sum_i \alpha_i v_i$ update — the worked example above, made live.

Self-attention

Queries, keys and values from one sequence

D DERIVATION

In **self**-attention, all three come from the **same** input X — each token is a query, a key, **and** a value:

Queries, keys and values from one sequence

D DERIVATION

In **self**-attention, all three come from the **same** input X — each token is a query, a key, **and** a value:

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

Queries, keys and values from one sequence

D DERIVATION

In **self**-attention, all three come from the **same** input X — each token is a query, a key, **and** a value:

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

Each token builds a **new representation** by mixing information from the other tokens it finds relevant — a query looks around, keys advertise, values are gathered.

Queries, keys and values from one sequence

D DERIVATION

In **self**-attention, all three come from the **same** input X — each token is a query, a key, **and** a value:

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

Each token builds a **new representation** by mixing information from the other tokens it finds relevant — a query looks around, keys advertise, values are gathered.

same operation as before, but the sequence attends **to itself**

The attention matrix

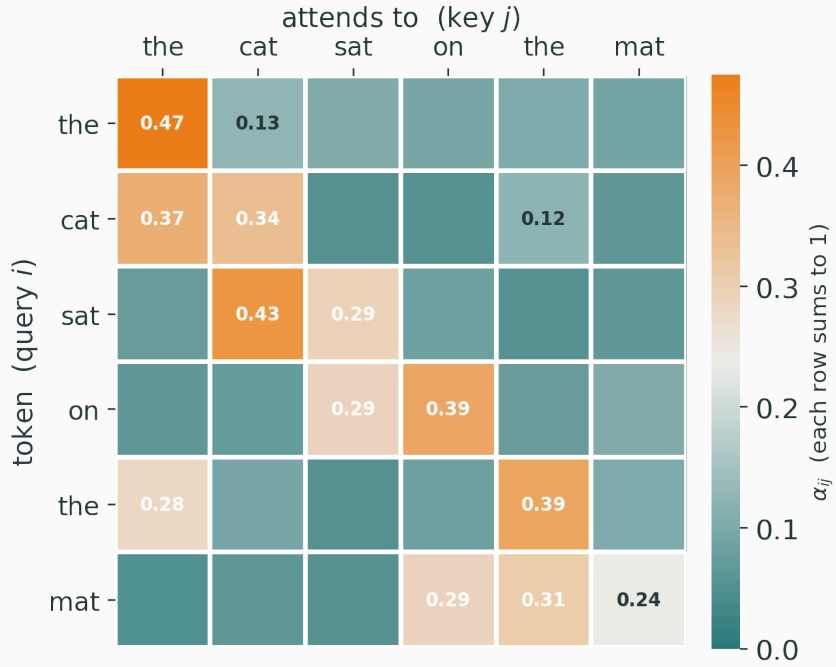
V VISUAL

A_{ij} = how much token i (query) attends to token j (key); every **row sums to 1**:

The attention matrix

V VISUAL

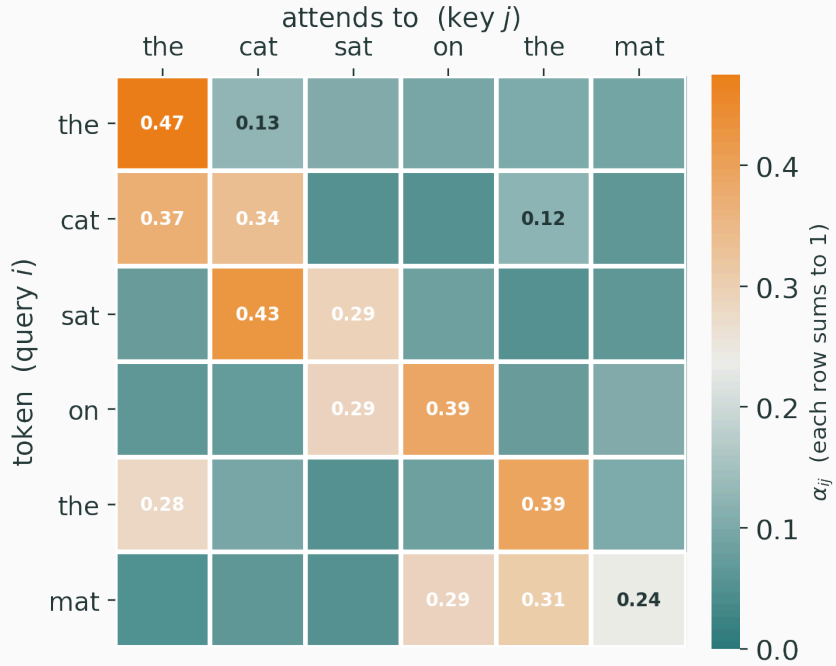
A_{ij} = how much token i (query) attends to token j (key); every **row sums to 1**:



The attention matrix

V VISUAL

A_{ij} = how much token i (query) attends to token j (key); every **row sums to 1**:



a $T \times T$ matrix — each row is a distribution over which tokens to gather from

Worked self-attention

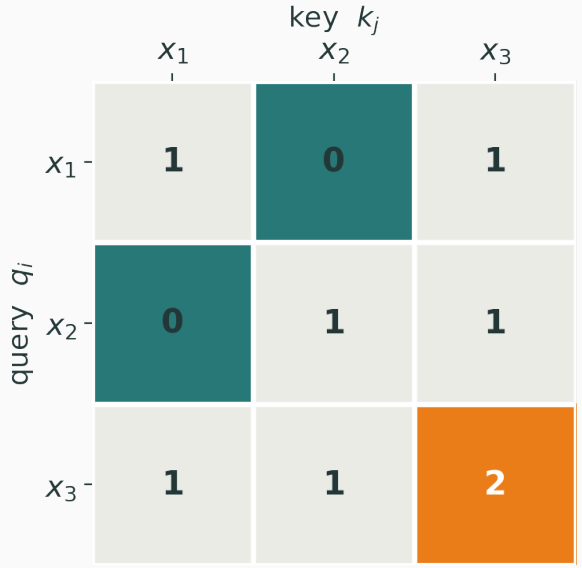
D DERIVATION

Take $W_Q = W_K = W_V = I$, so $Q = K = V = X$ with $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$. The scores $QK^\top = XX^\top$:

Worked self-attention

D DERIVATION

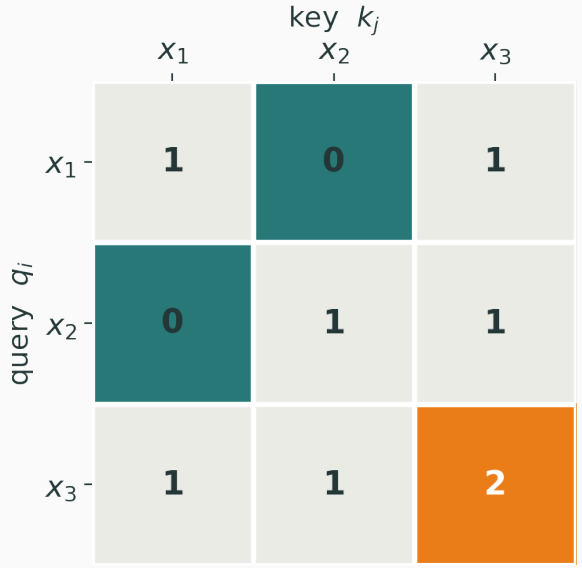
Take $W_Q = W_K = W_V = I$, so $Q = K = V = X$ with $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$. The scores $QK^\top = XX^\top$:



Worked self-attention

D DERIVATION

Take $W_Q = W_K = W_V = I$, so $Q = K = V = X$ with $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$. The scores $QK^\top = XX^\top$:



the diagonal grows with each token's norm; scale by $\sqrt{d_k} = \sqrt{2}$ before the softmax next

Reading a row

D DERIVATION

Scale by $\sqrt{2}$; row 1 (token 1 as query) before softmax is $(1 \;\; 0 \;\; 1)/\sqrt{2} = (0.71 \;\; 0 \;\; 0.71)$:

Reading a row

D DERIVATION

Scale by $\sqrt{2}$; row 1 (token 1 as query) before softmax is $(1 \;\; 0 \;\; 1)/\sqrt{2} = (0.71 \;\; 0 \;\; 0.71)$:

$$\alpha_1 = \mathrm{softmax}(0.71, 0, 0.71) \approx (0.40, 0.20, 0.40)$$

Reading a row

D DERIVATION

Scale by $\sqrt{2}$; row 1 (token 1 as query) before softmax is $(1 \;\; 0 \;\; 1)/\sqrt{2} = (0.71 \;\; 0 \;\; 0.71)$:

$$\alpha_1 = \mathrm{softmax}(0.71, 0, 0.71) \approx (0.40, 0.20, 0.40)$$

$$z_1 = 0.40\, x_1 + 0.20\, x_2 + 0.40\, x_3 \approx (0.80 \;\; 0.60)$$

Reading a row

D DERIVATION

Scale by $\sqrt{2}$; row 1 (token 1 as query) before softmax is $(1 \;\; 0 \;\; 1)/\sqrt{2} = (0.71 \;\; 0 \;\; 0.71)$:

$$\alpha_1 = \mathrm{softmax}(0.71, 0, 0.71) \approx (0.40, 0.20, 0.40)$$

$$z_1 = 0.40\, x_1 + 0.20\, x_2 + 0.40\, x_3 \approx (0.80 \;\; 0.60)$$

token 1 attends to tokens 1 and 3 more than 2 — its output is a weighted average of their values

Self-attention is content-dependent

D DERIVATION

Compare how a CNN and self-attention decide **which positions** interact:

Self-attention is content-dependent

D DERIVATION

Compare how a CNN and self-attention decide **which positions** interact:

CNN: the receptive field is **fixed by the kernel** — position i always mixes with $i-1, i, i+1$, regardless of content.

Self-attention: A_{ij} depends on the **token representations** — each token **decides** which others matter, per input.

Self-attention is content-dependent

D DERIVATION

Compare how a CNN and self-attention decide **which positions** interact:

CNN: the receptive field is **fixed by the kernel** — position i always mixes with $i-1, i, i+1$, regardless of content.

Self-attention: A_{ij} depends on the **token representations** — each token **decides** which others matter, per input.

same weights, different wiring for every sentence — the mixing pattern is learned and dynamic

Interactive: the self-attention matrix

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Edit a short sentence and watch the $T \times T$ attention matrix light up — each row a softmax distribution over which tokens that position pulls information from.

Masked (causal) self-attention

The future-leakage problem

D DERIVATION

For a **language model**, position t must predict x_{t+1} from **only** the past:

The future-leakage problem

D DERIVATION

For a **language model**, position t must predict x_{t+1} from **only** the past:

$$p(x_{t+1} \mid x_{\leq t})$$ (it may look at $x_1, \ldots, x_t$ — never ahead)

The future-leakage problem

D DERIVATION

For a **language model**, position t must predict x_{t+1} from **only** the past:

$p(x_{t+1} \mid x_{\leq t})$ (it may look at $x_1, \ldots, x_t$ — never ahead)

Plain self-attention lets token t attend to **every** position, including $x_{t+1}, x_{t+2}, \ldots$ — it would peek at the answer. That is **illegal** for autoregressive generation.

The causal mask

D DERIVATION

Compute the scores, then set the **future** entries to $-\infty$ before the softmax:

The causal mask

D DERIVATION

Compute the scores, then set the **future** entries to $-\infty$ before the softmax:

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad S_{ij} \leftarrow -\infty \;\text{ if } j > i$$

The causal mask

D DERIVATION

Compute the scores, then set the **future** entries to $-\infty$ before the softmax:

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad S_{ij} \leftarrow -\infty \;\text{ if } j > i$$

Softmax turns $-\infty$ into a weight of 0:

$$A_{ij} = 0 \quad \text{for all} \quad j > i$$

The causal mask

D DERIVATION

Compute the scores, then set the **future** entries to $-\infty$ before the softmax:

$$S = \frac{QK^\top}{\sqrt{d_k}}, \qquad S_{ij} \leftarrow -\infty \;\text{ if } j > i$$

Softmax turns $-\infty$ into a weight of 0:

$$A_{ij} = 0 \quad \text{for all} \quad j > i$$

each row now normalizes over the **past and present only** — the future contributes nothing

The causal mask, drawn

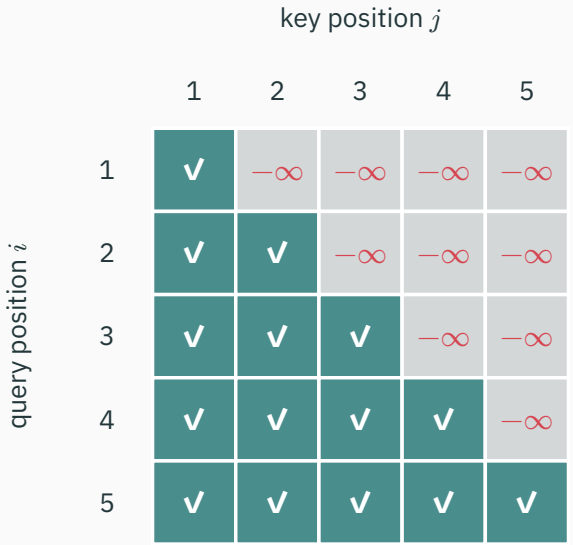
V VISUAL

$T = 5$: allowed entries (lower triangle) vs masked future ($-\infty$, upper triangle):

The causal mask, drawn

V VISUAL

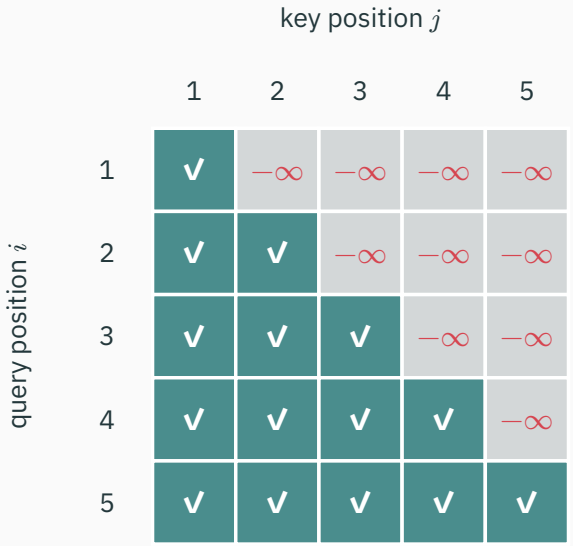
$T = 5$: allowed entries (lower triangle) vs masked future ($-\infty$, upper triangle):



The causal mask, drawn

V VISUAL

$T = 5$: allowed entries (lower triangle) vs masked future ($-\infty$, upper triangle):



query i may attend to key j only when $j \leq i$ — a lower-triangular attention pattern

Encoder vs decoder attention

D DERIVATION

The **same** self-attention op, two masking regimes:

Encoder vs decoder attention

D DERIVATION

The **same** self-attention op, two masking regimes:

Setting	**Mask**	**Sees**
encoder (BERT-style)	none	**bidirectional** — the whole sequence
decoder (GPT-style)	causal	only the past $x_{\leq t}$

Encoder vs decoder attention

D DERIVATION

The **same** self-attention op, two masking regimes:

Setting	**Mask**	**Sees**
encoder (BERT-style)	none	**bidirectional** — the whole sequence
decoder (GPT-style)	causal	only the past $x_{\leq t}$

bidirectional is great for **understanding**; causal is required for **generation**

Interactive: causal attention

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Toggle the causal mask on a short sequence and watch the upper triangle of the attention matrix go dark — the future becomes unreachable, one row at a time.

Multi-head attention

Why more than one head?

D DERIVATION

A single attention pattern can track **one** kind of relation at a time. Use H of them in parallel:

Why more than one head?

D DERIVATION

A single attention pattern can track **one** kind of relation at a time. Use H of them in parallel:

$$\text{head}_i = \text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

Why more than one head?

D DERIVATION

A single attention pattern can track **one** kind of relation at a time. Use H of them in parallel:

$$\text{head}_i = \text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

$$\text{MHA}(X) = \begin{pmatrix} \text{head}_1 \\ \vdots \\ \text{head}_H \end{pmatrix} W_O \qquad \text{(concatenate, then mix with } W_O\text{)}$$

Why more than one head?

D DERIVATION

A single attention pattern can track **one** kind of relation at a time. Use H of them in parallel:

$$\text{head}_i = \text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)$$

$$\text{MHA}(X) = \begin{pmatrix} \text{head}_1 \\ \vdots \\ \text{head}_H \end{pmatrix} W_O \qquad \text{(concatenate, then mix with } W_O\text{)}$$

each head projects into its **own** subspace and learns a **different** relation

Multi-head shapes

V VISUAL

The classic setup: $d_{\text{model}} = 512$, $H = 8$ heads:

Multi-head shapes

V VISUAL

The classic setup: $d_{\text{model}} = 512$, $H = 8$ heads:

$$d_k = d_v = \frac{d_{\text{model}}}{H} = \frac{512}{8} = 64$$

Multi-head shapes

V VISUAL

The classic setup: $d_{\text{model}} = 512$, $H = 8$ heads:

$$d_k = d_v = \frac{d_{\text{model}}}{H} = \frac{512}{8} = 64$$

Per head	Shape	Note
W_i^Q, W_i^K, W_i^V	512×64	projects into a 64-dim subspace
head_i	$T \times 64$	attention within that subspace
concat of 8 heads	$T \times 512$	back to full width
W_O	512×512	mixes the heads

What can heads learn?

D DERIVATION

Different heads specialize — some interpretable patterns observed in trained models:

What can heads learn?

D DERIVATION

Different heads specialize — some interpretable patterns observed in trained models:

- attend to the **previous** word
- **subject–verb** agreement
- matching **brackets** / quotes

- local **phrase** structure
- **positional** patterns
- coarse **semantic** similarity

What can heads learn?

D DERIVATION

Different heads specialize — some interpretable patterns observed in trained models:

- attend to the **previous** word
- **subject–verb** agreement
- matching **brackets** / quotes

- local **phrase** structure
- **positional** patterns
- coarse **semantic** similarity

Caution. Individual-head interpretation is **noisy** and post-hoc — heads are not cleanly labeled functions, and behavior is distributed across many of them.

Worked: multi-head parameter count

D DERIVATION

Combining all H heads, the per-projection weights stack into full $d \times d$ matrices. With $d = 512$:

Worked: multi-head parameter count

D DERIVATION

Combining all H heads, the per-projection weights stack into full $d \times d$ matrices. With $d = 512$:

$$W_Q, W_K, W_V, W_O \in \mathbb{R}^{512 \times 512}$$

Worked: multi-head parameter count

D DERIVATION

Combining all H heads, the per-projection weights stack into full $d \times d$ matrices. With $d = 512$:

$$W_Q, W_K, W_V, W_O \in \mathbb{R}^{512 \times 512}$$

$$4 \times 512^2 = 4 \times 262{,}144 =$$

$1{,}048{,}576$ parameters

Worked: multi-head parameter count

D DERIVATION

Combining all H heads, the per-projection weights stack into full $d \times d$ matrices. With $d = 512$:

$$W_Q, W_K, W_V, W_O \in \mathbb{R}^{512 \times 512}$$

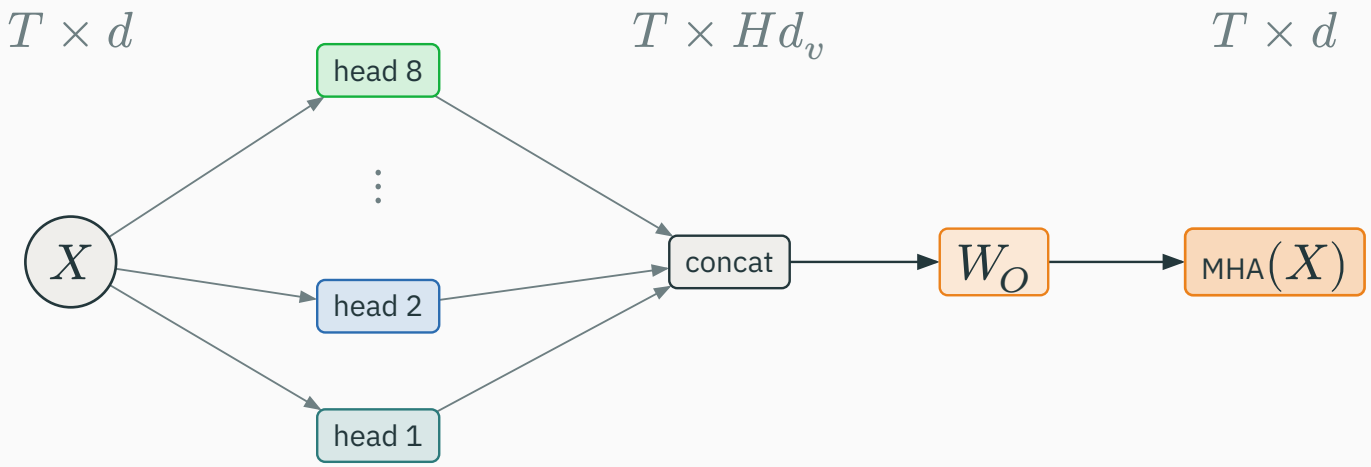
$$4 \times 512^2 = 4 \times 262{,}144 =$$

$1{,}048{,}576$ parameters

≈ 1.05 M — this is one attention sublayer; a $512 \to 2048 \to 512$ FFN adds ≈ 2.1 M, about twice as much

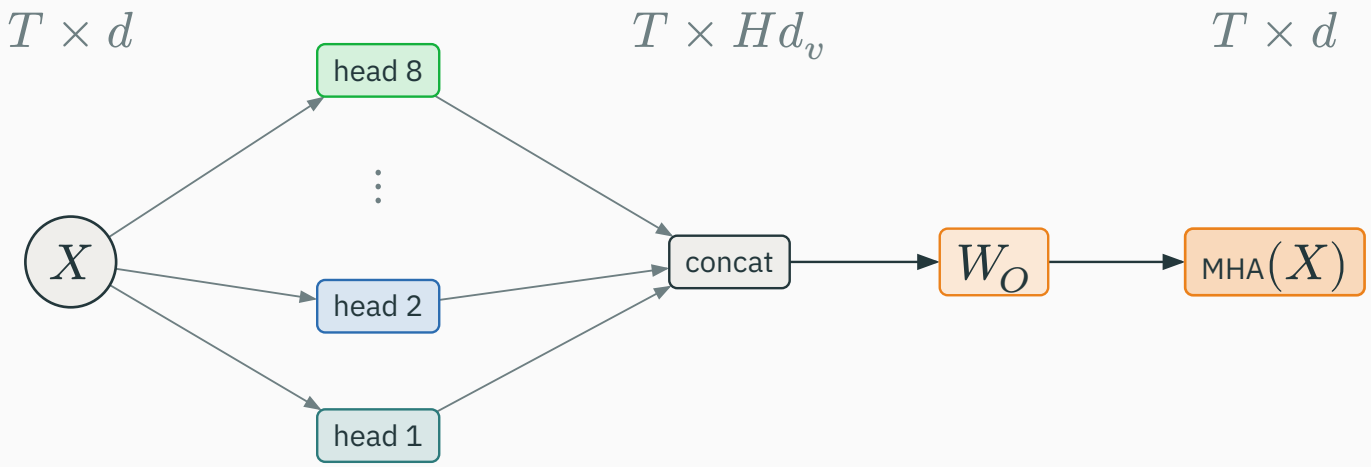
The multi-head block

V VISUAL



The multi-head block

V VISUAL



H attentions run in parallel on the same X, concatenate, then W_O mixes them back to width d

Interactive: multi-head attention

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Switch between heads and see each one attend to a **different** relation on the same sentence — then watch their outputs concatenate and mix through W_O.

Positional information

Self-attention forgets order

D DERIVATION

Attention is a **weighted sum** — it depends on **content**, not position. Shuffle the tokens and:

Self-attention forgets order

D DERIVATION

Attention is a **weighted sum** — it depends on **content**, not position. Shuffle the tokens and:

$$\mathrm{SelfAttention}\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right) \quad \text{is a permutation of} \quad \mathrm{SelfAttention}\left(\begin{pmatrix} x_2 \\ x_1 \\ x_3 \end{pmatrix}\right)$$

Self-attention forgets order

D DERIVATION

Attention is a **weighted sum** — it depends on **content**, not position. Shuffle the tokens and:

$$\text{SelfAttention}\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right) \quad \text{is a permutation of} \quad \text{SelfAttention}\left(\begin{pmatrix} x_2 \\ x_1 \\ x_3 \end{pmatrix}\right)$$

Self-attention is **permutation-equivariant**: with no position signal, the model cannot tell “dog bites man” from “man bites dog”. Order is invisible.

Self-attention forgets order

D DERIVATION

Attention is a **weighted sum** — it depends on **content**, not position. Shuffle the tokens and:

$$\text{SelfAttention}\left(\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}\right) \quad \text{is a permutation of} \quad \text{SelfAttention}\left(\begin{pmatrix} x_2 \\ x_1 \\ x_3 \end{pmatrix}\right)$$

Self-attention is **permutation-equivariant**: with no position signal, the model cannot tell “dog bites man” from “man bites dog”. Order is invisible.

we must inject position information into the token representations

Adding a positional encoding

D DERIVATION

Give every position t a vector p_t and **add** it to the token embedding:

Adding a positional encoding

D DERIVATION

Give every position t a vector p_t and **add** it to the token embedding:

$$r_t = e_t + p_t \qquad \text{(token content + where it sits)}$$

Adding a positional encoding

D DERIVATION

Give every position t a vector p_t and **add** it to the token embedding:

$$r_t = e_t + p_t \qquad \text{(token content + where it sits)}$$

Now two identical tokens at different positions get **different** inputs r_t — attention can use the position signal, and permuting the sequence changes the output.

Learned positional embeddings

D DERIVATION

The simplest choice: a **lookup table** of position vectors, trained like any embedding:

Learned positional embeddings

D DERIVATION

The simplest choice: a **lookup table** of position vectors, trained like any embedding:

$$P \in \mathbb{R}^{T_{\max} \times d}, \qquad p_t = P_{t,:}$$

Learned positional embeddings

D DERIVATION

The simplest choice: a **lookup table** of position vectors, trained like any embedding:

$$P \in \mathbb{R}^{T_{\max} \times d}, \qquad p_t = P_{t,:}$$

Pro — simple, flexible, learns whatever the data needs (GPT / BERT use this).

Con — undefined for $t > T_{\max}$; cannot extrapolate to sequences longer than seen in training.

Sinusoidal positional encoding

D DERIVATION

The original Transformer uses a **fixed**, deterministic pattern of sines and cosines:

Sinusoidal positional encoding

D DERIVATION

The original Transformer uses a **fixed**, deterministic pattern of sines and cosines:

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

Sinusoidal positional encoding

D DERIVATION

The original Transformer uses a **fixed**, deterministic pattern of sines and cosines:

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

Sinusoidal positional encoding

D DERIVATION

The original Transformer uses a **fixed**, deterministic pattern of sines and cosines:

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

dimension $2i$ oscillates at wavelength $2\pi \cdot 10000^{2i/d}$ — no parameters, defined for **any** position

Many frequencies, many scales

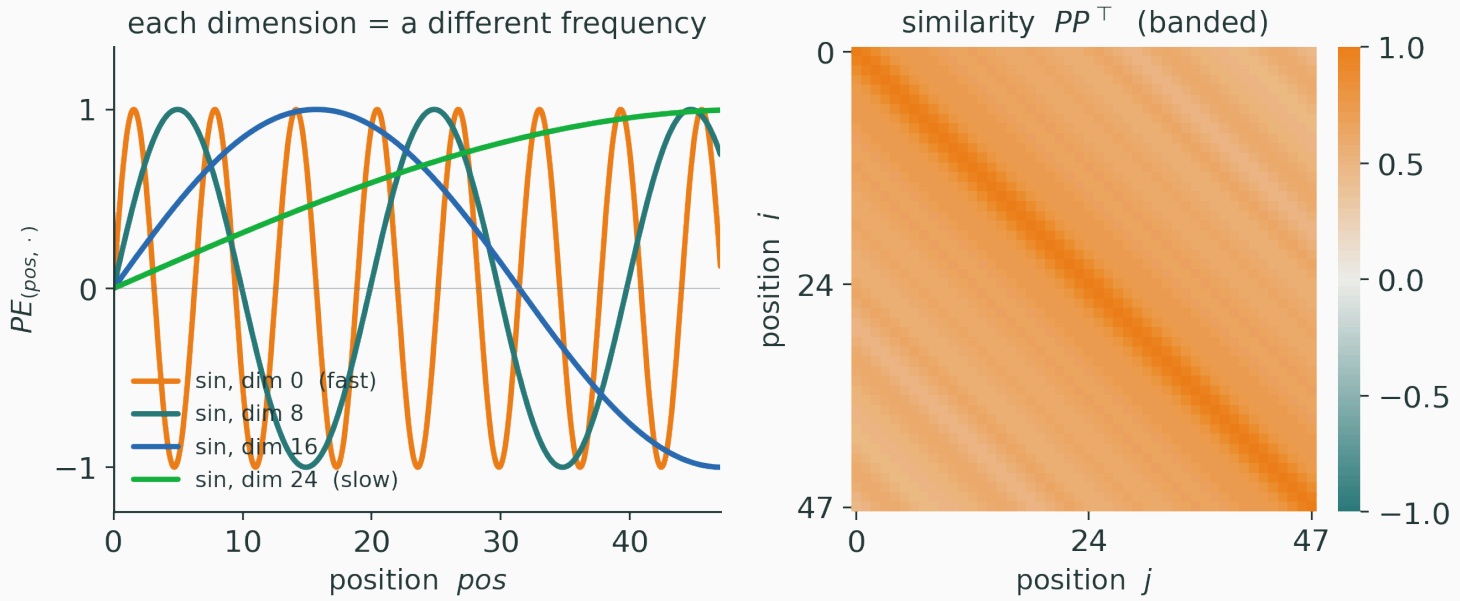
V VISUAL

Low dimensions oscillate **fast**, high dimensions **slow** — position encoded at every scale:

Many frequencies, many scales

V VISUAL

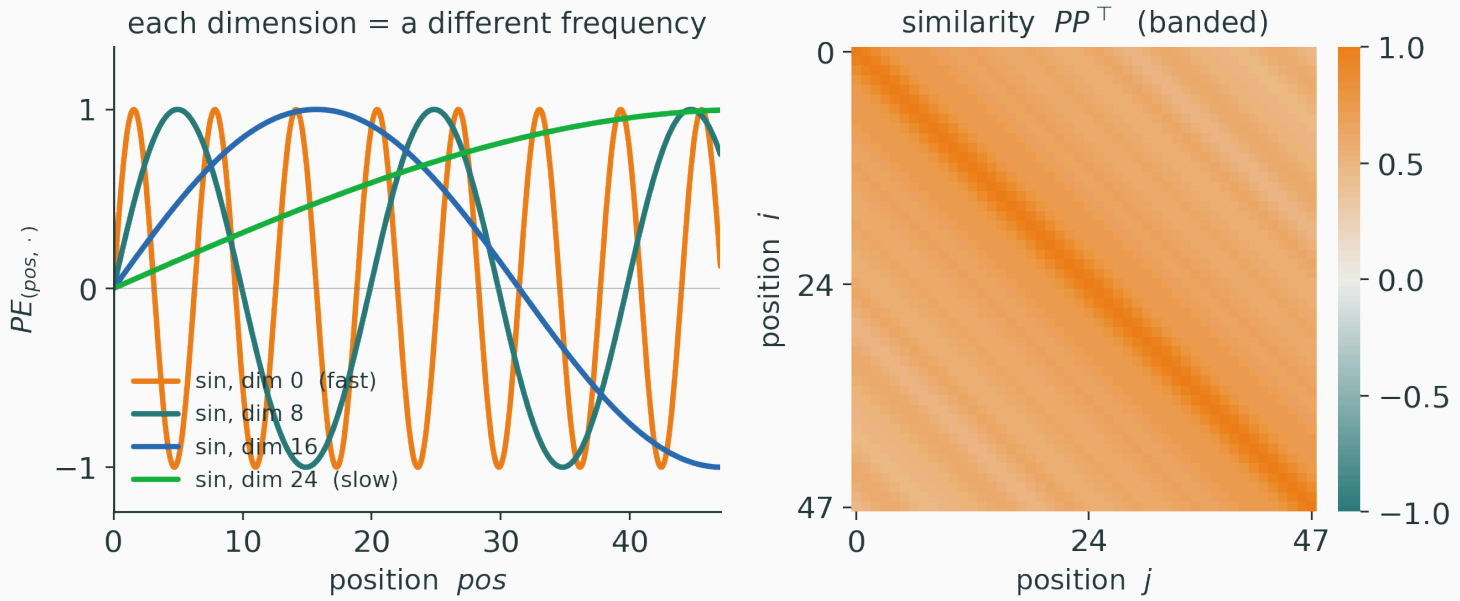
Low dimensions oscillate **fast**, high dimensions **slow** — position encoded at every scale:



Many frequencies, many scales

V VISUAL

Low dimensions oscillate **fast**, high dimensions **slow** — position encoded at every scale:



left: sinusoids at different frequencies · right: $PP^\top$ — nearby positions have **similar** encodings

Worked position vector

D DERIVATION

Take $d = 4$, so $i \in \{0, 1\}$ and the exponents are $10000^{0} = 1$ and $10000^{1/2} = 100$:

Worked position vector

D DERIVATION

Take $d = 4$, so $i \in \{0, 1\}$ and the exponents are $10000^0 = 1$ and $10000^{1/2} = 100$:

$$\mathrm{PE}(\mathrm{pos}) = \left(\sin\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \sin\left(\tfrac{\mathrm{pos}}{100}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{100}\right)\right)$$

Worked position vector

D DERIVATION

Take $d = 4$, so $i \in \{0, 1\}$ and the exponents are $10000^{0} = 1$ and $10000^{1/2} = 100$:

$$\mathrm{PE}(\mathrm{pos}) = \left(\sin\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \sin\left(\tfrac{\mathrm{pos}}{100}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{100}\right)\right)$$

At $\mathrm{pos} = 0$ every sine is 0 and every cosine is 1:

$$\mathrm{PE}(0) = (0 \;\; 1 \;\; 0 \;\; 1)$$

Worked position vector

D DERIVATION

Take $d = 4$, so $i \in \{0, 1\}$ and the exponents are $10000^{0} = 1$ and $10000^{1/2} = 100$:

$$\mathrm{PE}(\mathrm{pos}) = \left(\sin\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{1}\right) \;\; \sin\left(\tfrac{\mathrm{pos}}{100}\right) \;\; \cos\left(\tfrac{\mathrm{pos}}{100}\right)\right)$$

At $\mathrm{pos} = 0$ every sine is 0 and every cosine is 1:

$$\mathrm{PE}(0) = (0 \;\; 1 \;\; 0 \;\; 1)$$

a unique, smooth signature per position — added onto the token embedding

Interactive: positional encoding

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/positional-encoding

Slide the position and watch each sinusoidal dimension turn; compare learned vs sinusoidal encodings and see how nearby positions stay similar while distant ones drift apart.

The Transformer block

The components

D DERIVATION

One block stacks two sublayers, each wrapped in a residual connection and a normalization:

The components

D DERIVATION

One block stacks two sublayers, each wrapped in a residual connection and a normalization:

$$\text{multi-head self-attention} \to \text{residual} + \text{norm} \to \text{FFN} \to \text{residual} + \text{norm}$$

The components

D DERIVATION

One block stacks two sublayers, each wrapped in a residual connection and a normalization:

$$\text{multi-head self-attention} \to \text{residual} + \text{norm} \to \text{FFN} \to \text{residual} + \text{norm}$$

Attention mixes information **across positions**; the **FFN** then processes **each position** on its own. Communication, then computation.

The feed-forward network

D DERIVATION

Applied **independently and identically** to every position — a two-layer MLP:

The feed-forward network

D DERIVATION

Applied **independently and identically** to every position — a two-layer MLP:

$$\mathrm{FFN}(x) = W_2\,\varphi(W_1 x + b_1) + b_2$$

The feed-forward network

D DERIVATION

Applied **independently and identically** to every position — a two-layer MLP:

$$\mathrm{FFN}(x) = W_2\,\varphi(W_1 x + b_1) + b_2$$

- same weights W_1, W_2 at **every** position (a 1×1 transform over the sequence);
- original sizing: $d_{\text{model}} = 512$ expands to $d_{\text{ff}} = 2048$, then back to 512.

The feed-forward network

D DERIVATION

Applied **independently and identically** to every position — a two-layer MLP:

$$\mathrm{FFN}(x) = W_2\,\varphi(W_1 x + b_1) + b_2$$

- same weights W_1, W_2 at **every** position (a 1×1 transform over the sequence);
- original sizing: $d_{\text{model}} = 512$ expands to $d_{\text{ff}} = 2048$, then back to 512.

φ is usually ReLU or GELU; the hidden layer is $4\times$ wider than the model

Communication, then computation

D DERIVATION

Why an FFN **after** attention? The two sublayers do **complementary** jobs:

Communication, then computation

D DERIVATION

Why an FFN **after** attention? The two sublayers do **complementary** jobs:

Attention = **communication**. Tokens exchange information — each position gathers from others.

FFN = **computation**. Each token, now informed, is transformed on its own — no mixing.

Communication, then computation

D DERIVATION

Why an FFN **after** attention? The two sublayers do **complementary** jobs:

Attention = **communication**. Tokens exchange information — each position gathers from others.

FFN = **computation**. Each token, now informed, is transformed on its own — no mixing.

attention decides **what to gather**; the FFN decides **what to do** with it, tokenwise

Residuals and normalization

D DERIVATION

Each sublayer is wrapped so gradients flow and activations stay well-scaled:

Residuals and normalization

D DERIVATION

Each sublayer is wrapped so gradients flow and activations stay well-scaled:

post-norm : $x' = \mathrm{LN}(x + \mathrm{MHA}(x)), \quad y = \mathrm{LN}(x' + \mathrm{FFN}(x'))$

Residuals and normalization

D DERIVATION

Each sublayer is wrapped so gradients flow and activations stay well-scaled:

post-norm : $x' = \mathrm{LN}(x + \mathrm{MHA}(x)), \quad y = \mathrm{LN}(x' + \mathrm{FFN}(x'))$

pre-norm : $x' = x + \mathrm{MHA}(\mathrm{LN}(x)), \quad y = x' + \mathrm{FFN}(\mathrm{LN}(x'))$

Residuals and normalization

D DERIVATION

Each sublayer is wrapped so gradients flow and activations stay well-scaled:

post-norm : $x' = \mathrm{LN}(x + \mathrm{MHA}(x)), \quad y = \mathrm{LN}(x' + \mathrm{FFN}(x'))$

pre-norm : $x' = x + \mathrm{MHA}(\mathrm{LN}(x)), \quad y = x' + \mathrm{FFN}(\mathrm{LN}(x'))$

pre-norm keeps a clean residual highway — more stable for **deep** stacks (the modern default)

The Transformer encoder block

V VISUAL

the horizontal spine is the **residual stream** (LN = LayerNorm)

The Transformer encoder block

V VISUAL

the horizontal spine is the **residual stream** (LN = LayerNorm)

the residual stream runs straight through; each sublayer **branches off, computes, and adds back**

The decoder block

D DERIVATION

A full encoder–decoder decoder layer has **three** sublayers:

The decoder block

D DERIVATION

A full encoder–decoder decoder layer has **three** sublayers:

1. **masked** self-attention over the output so far (causal);
2. **cross-attention** — queries from the decoder, keys and values from the **encoder**;
3. a position-wise FFN.

The decoder block

D DERIVATION

A full encoder–decoder decoder layer has **three** sublayers:

1. **masked** self-attention over the output so far (causal);
2. **cross-attention** — queries from the decoder, keys and values from the **encoder**;
3. a position-wise FFN.

cross-attention is how the decoder reads the source — we unpack encoder–decoder next lecture

The decoder-only (GPT-style) block

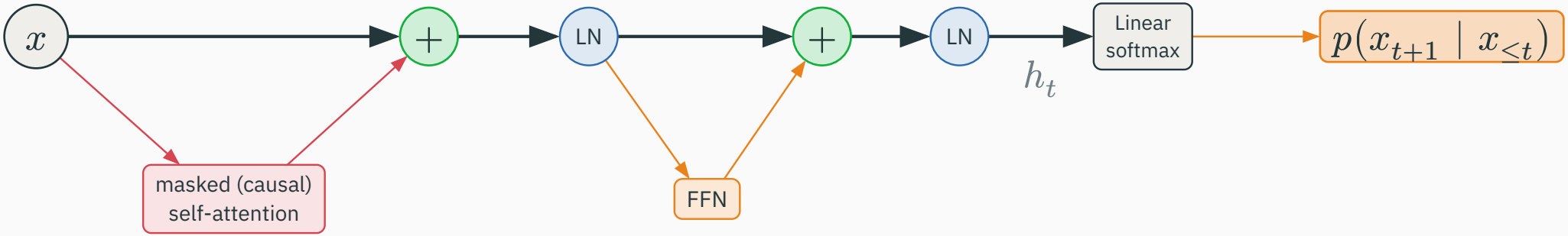
V VISUAL

Drop the encoder and cross-attention: **causal self-attention + FFN**, then an LM head:

The decoder-only (GPT-style) block

V VISUAL

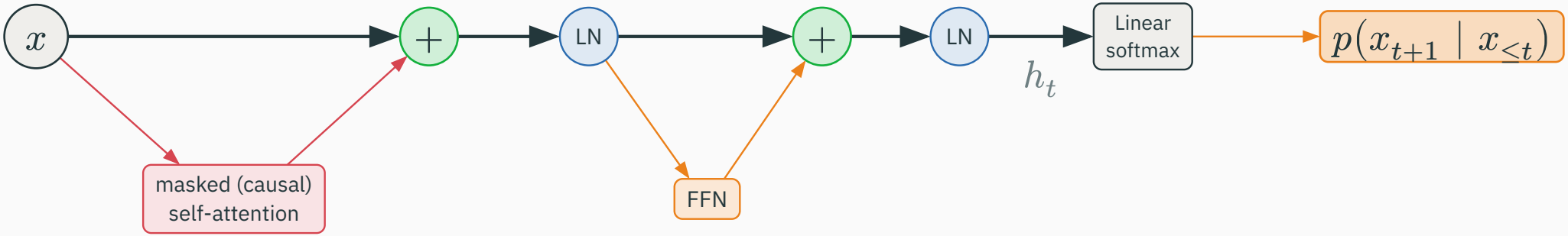
Drop the encoder and cross-attention: **causal self-attention + FFN**, then an LM head:



The decoder-only (GPT-style) block

V VISUAL

Drop the encoder and cross-attention: **causal self-attention + FFN**, then an LM head:



output $p(x_{t+1} \mid x_{\le t}) = \mathrm{softmax}(W_o\, h_t)$ – this is the GPT family

Interactive: the Transformer block

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/transformer-block

Send a sequence through one block and watch the residual stream: attention mixes across positions, the FFN transforms each, and the norms keep it stable.

Complexity and comparison

The cost of self-attention

D DERIVATION

The attention matrix is $T \times T$ — quadratic in sequence length:

The cost of self-attention

D DERIVATION

The attention matrix is $T \times T$ — quadratic in sequence length:

$$QK^\top : \quad O(T^2 d) \text{ compute}, \qquad A \in \mathbb{R}^{T \times T} : \quad O(T^2) \text{ memory}$$

The cost of self-attention

D DERIVATION

The attention matrix is $T \times T$ — quadratic in sequence length:

$$QK^\top : \quad O(T^2 d) \text{ compute}, \qquad A \in \mathbb{R}^{T \times T} : \quad O(T^2) \text{ memory}$$

Every token attends to every token, so cost grows with T^2. This is the **long-context bottleneck** — the price of full-range, content-dependent mixing.

The cost of self-attention

D DERIVATION

The attention matrix is $T \times T$ — quadratic in sequence length:

$$QK^\top : \quad O(T^2 d) \text{ compute}, \qquad A \in \mathbb{R}^{T \times T} : \quad O(T^2) \text{ memory}$$

Every token attends to every token, so cost grows with T^2. This is the **long-context bottleneck** — the price of full-range, content-dependent mixing.

fine for $T \sim 10^3$; the target of much later work (efficient / sparse attention) for very long T

Path length: how far is information?

D DERIVATION

How many steps must a signal travel between positions i and j?

Path length: how far is information?

D DERIVATION

How many steps must a signal travel between positions i and j?

Model	**Path length $i \to j$**
RNN	$O(\lvert i-j \rvert)$ — along the chain
dilated CNN	$O(\log \lvert i-j \rvert)$ — up the dilation tree
self-attention	$O(1)$ — one hop, any distance

Path length: how far is information?

D DERIVATION

How many steps must a signal travel between positions i and j?

Model	Path length $i \to j$
RNN	$O(\lvert i-j \rvert)$ — along the chain
dilated CNN	$O(\log \lvert i-j \rvert)$ — up the dilation tree
self-attention	$O(1)$ — one hop, any distance

short paths = easy gradient flow = long-range dependencies are learnable

Model comparison

V VISUAL

	RNN	TCN (CNN)	Transformer
parallel over time	low	high	high
longest path $i \to j$	$O(T)$	$O(\log T)$	$O(1)$
compute per layer	$O(Td^2)$	$O(Tkd^2)$	$O(T^2d)$
context	state (decays)	fixed field k	content-dependent, full

Model comparison

V VISUAL

	RNN	TCN (CNN)	Transformer
parallel over time	low	high	high
longest path $i \to j$	$O(T)$	$O(\log T)$	$O(1)$
compute per layer	$O(Td^2)$	$O(Tkd^2)$	$O(T^2d)$
context	state (decays)	fixed field k	content-dependent, full

the Transformer trades $O(T^2)$ cost for **parallelism** and **constant** path length

Why Transformers won

D DERIVATION

Why Transformers won

D DERIVATION

- **parallel training** — no sequential dependency, the whole sequence at once;

Why Transformers won

D DERIVATION

- **parallel training** — no sequential dependency, the whole sequence at once;
- **dynamic context** — content-dependent attention, not a fixed window;

Why Transformers won

D DERIVATION

- **parallel training** — no sequential dependency, the whole sequence at once;
- **dynamic context** — content-dependent attention, not a fixed window;
- **short paths** — $O(1)$ between any two positions, so long-range signals survive;

Why Transformers won

D DERIVATION

- **parallel training** — no sequential dependency, the whole sequence at once;
- **dynamic context** — content-dependent attention, not a fixed window;
- **short paths** — $O(1)$ between any two positions, so long-range signals survive;
- **stackable** — residual $+$ norm blocks compose into very deep, scalable models.

Why Transformers won

D DERIVATION

- **parallel training** — no sequential dependency, the whole sequence at once;
- **dynamic context** — content-dependent attention, not a fixed window;
- **short paths** — $O(1)$ between any two positions, so long-range signals survive;
- **stackable** — residual $+$ norm blocks compose into very deep, scalable models.

the same architecture scales from a toy LM to the largest models trained today

A tiny decoder-only language model

The whole pipeline

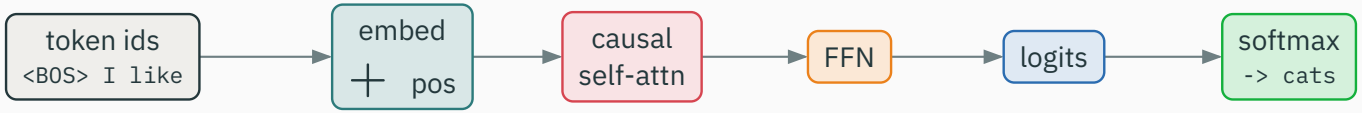
V VISUAL

A minimal GPT: turn `<BOS> I like` into a prediction of the next token:

The whole pipeline

V VISUAL

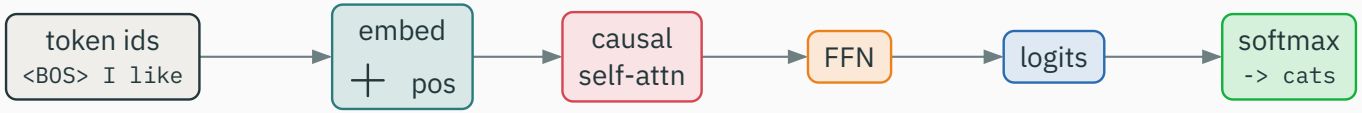
A minimal GPT: turn `<BOS> I like` into a prediction of the next token:



The whole pipeline

V VISUAL

A minimal GPT: turn `<BOS> I like` into a prediction of the next token:



token ids ⟶ embeddings + position ⟶ causal self-attention ⟶ FFN ⟶ logits ⟶ softmax

The training loss

D DERIVATION

The logits at position t predict the token at $t+1$ — the **same next-token objective** as before:

The training loss

D DERIVATION

The logits at position t predict the token at $t+1$ — the **same next-token objective** as before:

$$\mathcal{L} = -\sum_{t=1}^{T-1} \log p_\theta(x_{t+1} \mid x_{\leq t})$$

The training loss

D DERIVATION

The logits at position t predict the token at $t+1$ — the **same next-token objective** as before:

$$\mathcal{L} = -\sum_{t=1}^{T-1} \log p_\theta(x_{t+1} \mid x_{\leq t})$$

Nothing new in the loss — token-level cross-entropy. What changed is **how** p_θ is computed: causal self-attention over the prefix, not a recurrent state.

Shifted targets

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

Shifted targets

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

position t	1	2	3	4
input x_t	<BOS>	I	like	cats
target x_{t+1}	I	like	cats	<EOS>

Shifted targets

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

position t	1	2	3	4
input x_t	<BOS>	I	like	cats
target x_{t+1}	I	like	cats	<EOS>

the **causal mask** is what makes this safe — position t cannot see its own target x_{t+1}

Interactive: tiny Transformer LM

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/decoding-lab

Type a short prefix and watch it flow through embeddings, causal attention and the FFN to a next-token distribution; sample a token and feed it back.

Summary

What changed from RNNs

D DERIVATION

RNN	**Transformer**
$h_t = f(h_{t-1}, x_t)$	$H' = \mathrm{SelfAttention}(H)$
memory carried through **state**	interactions computed **directly**
serial over time	parallel over positions
path $i \to j$ is $O(\lvert i - j \rvert)$	path $i \to j$ is $O(1)$

What changed from RNNs

D DERIVATION

RNN	**Transformer**
$h_t = f(h_{t-1}, x_t)$	$H' = \mathrm{SelfAttention}(H)$
memory carried through **state**	interactions computed **directly**
serial over time	parallel over positions
path $i \to j$ is $O(\lvert i - j \rvert)$	path $i \to j$ is $O(1)$

recurrence **routes** information step by step; attention **relates** all positions at once

The key equations

V VISUAL

Step	Equation
project	$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$
attend	$A = \mathrm{softmax}\left(QK^\top / \sqrt{d_k} + \mathrm{mask}\right)$
gather	$O = AV$
multi-head	$\mathrm{MHA} = \begin{pmatrix} \mathrm{head}_1 \\ \vdots \\ \mathrm{head}_H \end{pmatrix} W_O$
block	$\mathrm{MHSA} \to$ residual $+$ norm $\to$ FFN $\to$ residual $+$ norm

Final mental model — one idea

Self-attention: every token asks the others for information.

Multi-head: several relation types, in parallel.

Positional encoding: puts the order back in.

a Transformer block = attention + tokenwise MLP + residual / norm

We replaced recurrence with **attention between all tokens**.

Next — **Transformers II**: encoder-only (BERT), decoder-only (GPT), encoder–decoder; pretraining, finetuning and generation.