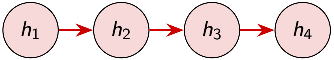
The Transformer Architecture

Self-Attention · Multi-Head Attention · Positional Encoding · Encoder–Decoder

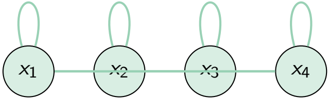
The key idea

LSTM



Sequential: $O(n)$ path

Transformer



Parallel: $O(1)$ path

“Attention Is All You Need” (Vaswani et al., 2017)

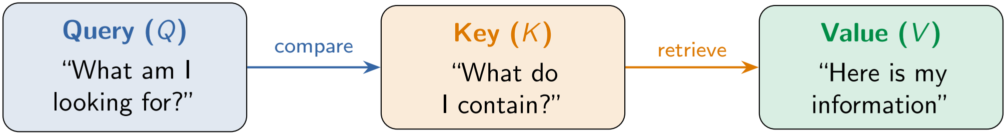
Parallel
Process all tokens simultaneously

Direct access
Any token can attend to any other

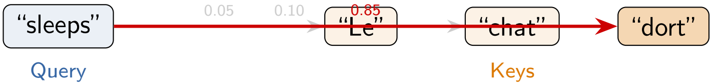
Contextual
Representations change with context

Attention — intuition

Think of it as a soft dictionary lookup

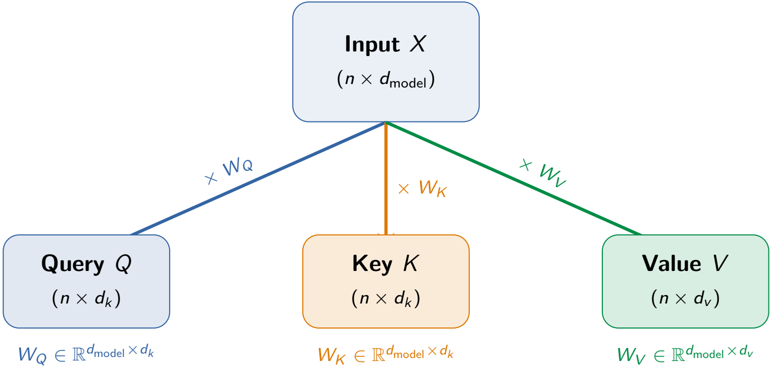


Example: translating "Le chat dort" → "The cat sleeps"



Output = weighted sum of values: $0.05 \cdot v_{\mathrm{Le}} + 0.10 \cdot v_{\mathrm{chat}} + 0.85 \cdot v_{\mathrm{dort}}$

Q, K, V — where do they come from?



Same input, three different views — the model **learns** what to query, what to advertise, and what to send

Self-attention: Q, K, V all from same X

Cross-attention: Q from decoder, K, V from encoder

Scaled dot-product attention

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

$QK^\top$ Similarity scores → $\div\sqrt{d_k}$ Scale down → softmax Normalize → $\times\ V$ Weighted sum

Why $\sqrt{d_k}$?

Without scaling, dot products grow with d_k, pushing softmax into saturation (near 0 or 1) where gradients vanish

Dimensions:

Q: $(n \times d_k)$ K: $(m \times d_k)$
V: $(m \times d_v)$
$QK^\top$: $(n \times m)$ Output: $(n \times d_v)$

Each output row is a **weighted average** of value vectors, where weights come from query-key similarity

Attention — worked example

Tokens: The cat sat

$QK^\top/\sqrt{d_k}$

	The	cat	sat
The	1.2	0.5	0.3
cat	0.4	2.1	0.8
sat	0.2	0.9	1.5

$\xrightarrow{\text{softmax}}$

Weights

.49	.28	.23
.12	.63	.25
.15	.30	.55

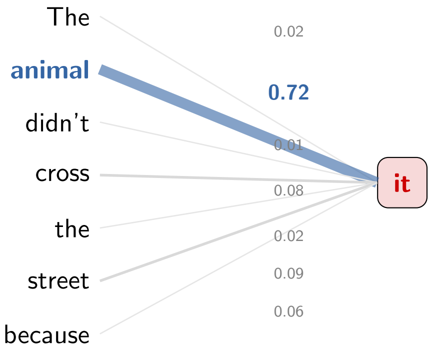
Row for "cat": attends 63% to itself, 25% to "sat", 12% to "The"

Output for "cat" $=$
$0.12 \cdot v_{\text{The}} + 0.63 \cdot v_{\text{cat}} + 0.25 \cdot v_{\text{sat}}$

Each output is a **context-aware** representation — unlike Word2Vec, the same word gets different embeddings in different contexts

Attention learns meaningful relationships

"The **animal** didn't cross the street because **it** was too tired"



Coreference resolved!

"it" attends most strongly to "animal" — the model learned that "it" refers to "animal"

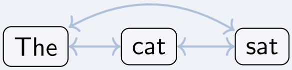
No explicit rule

This emerges from training — the model learns which tokens are relevant for each query

Different attention heads specialize in different relationships (see next slides)

Self-attention vs. cross-attention

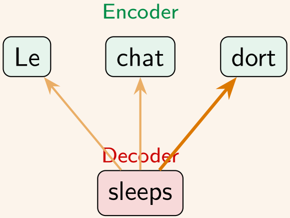
Self-Attention



Q, K, V all come from
the **same** sequence

Used in: encoder, decoder (masked)

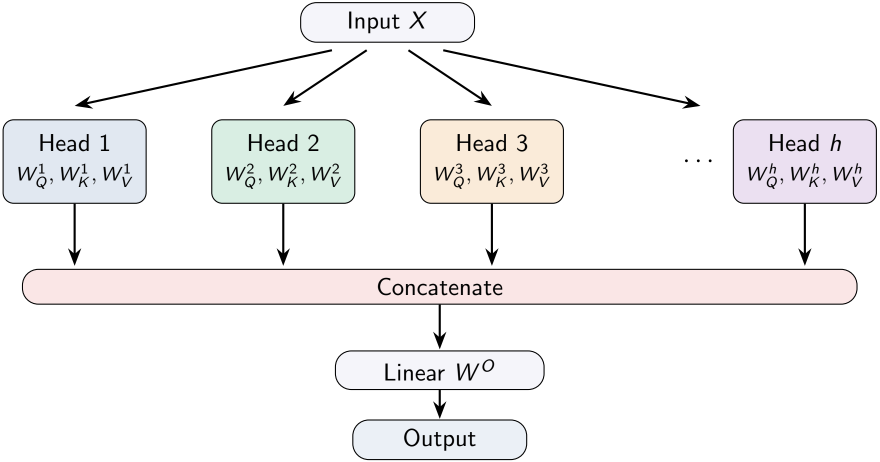
Cross-Attention



Q from decoder
K, V from encoder

Used in: decoder (enc-dec models)

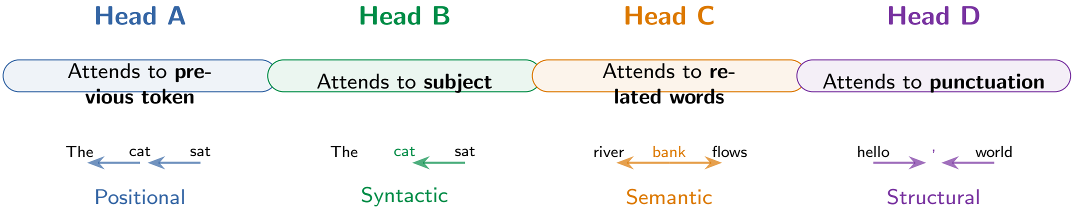
Multi-head attention



$$
\begin{aligned}
\mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, \dots, \mathrm{head}_h)\, W^O \\
\text{where } \mathrm{head}_i &= \mathrm{Attention}(QW_Q^i, KW_K^i, VW_V^i)
\end{aligned}
$$

What different heads learn

Different heads specialize in different types of relationships:



Multiple heads = multiple "perspectives" on the same input.

Typical: $h = 8$ (BERT-base) or $h = 12$–96 (larger models). $d_k = d_{\text{model}}/h$

Total compute is the same as single-head attention with full d_{model}, since each head uses $d_k = d_{\text{model}}/h$

Positional encoding — why we need it

“**dog** bites **man**”

Same tokens!

Problem: Self-attention computes dot products between token pairs.

It’s **permutation-invariant** — swapping token order doesn’t change the output!

But word order matters: these two sentences mean very different things.

Solution: Add **positional information** to the input embeddings

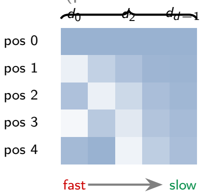
$$\text{input}_i \;=\; \text{token\_embedding}_i + \text{position\_encoding}_i$$

Sinusoidal positional encoding

$$
\begin{aligned}
\mathrm{PE}(\mathrm{pos}, 2i) &= \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right) \\
\mathrm{PE}(\mathrm{pos}, 2i + 1) &= \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)
\end{aligned}
$$

PE matrix

(position × dimension)



Unique pattern

Each position gets a distinct encoding

Relative positions

PE_{pos+k} can be expressed as linear function of PE_{pos}

No learned params

Fixed, deterministic, works for any length

Multi-scale

Low dims = fine position, high dims = coarse

$\mathrm{input}_i = \mathrm{embedding}(x_i) + \mathrm{PE}(i)$ (element-wise addition)

Modern positional encodings

Sinusoidal

Vaswani et al., 2017

Added to embeddings
Absolute position
Fixed (not learned)

Used by: original Transformer

RoPE

Su et al., 2021

Rotates Q and K vectors
Relative position
No extra parameters

Used by: LLaMA, Mistral

ALiBi

Press et al., 2022

Bias in attention scores
Linear distance penalty
No extra parameters

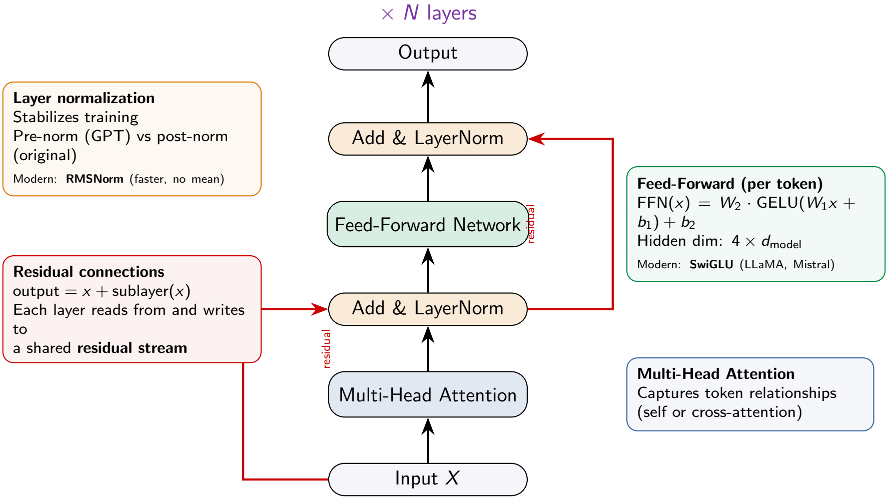
Used by: BLOOM, MPT

Where position info is injected:

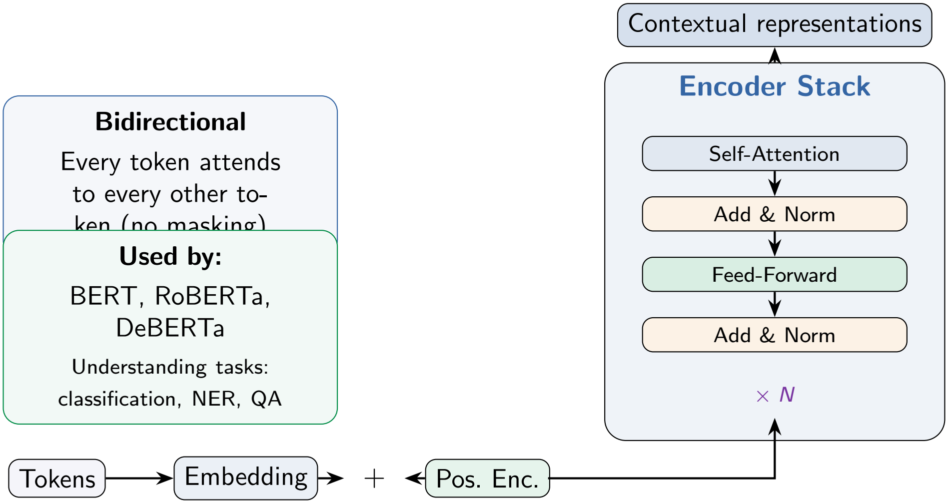
Sinusoidal → Embedding → RoPE → Q, K, V → ALiBi → $QK^\top$ → softmax → output

RoPE is the most popular today — it naturally encodes relative position and supports length extrapolation with techniques like YaRN

The Transformer block



The Transformer encoder



The Transformer decoder

From encoder
(K, V)

Decoder Block

Masked Self-Attention

Add & Norm

Cross-Attention

Add & Norm

Feed-Forward

Add & Norm

$\times N$

Causal Mask

attend masked

Token t can only
see tokens $\le t$

Decoder-only models (GPT) skip cross-attention — only masked self-attention + FFN

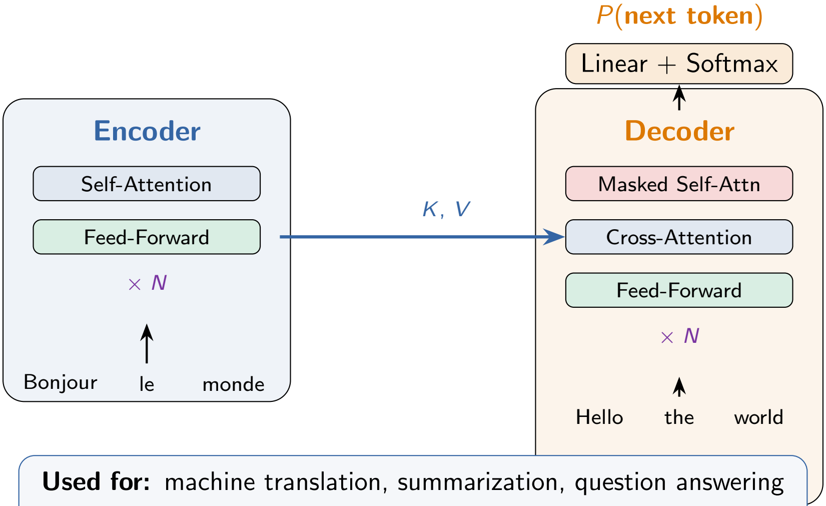
Used by:

GPT (decoder-only)
T5 decoder

Generation tasks

Dropout (p=0.1) applied after attention weights and after each sublayer during training

Encoder–Decoder architecture



Used for: machine translation, summarization, question answering

Models: T5, BART, mBART, original Transformer

Three Transformer architectures

Encoder-Only

Bidirectional
MLM training
Understanding tasks

BERT, RoBERTa

Decoder-Only

Autoregressive (L$\rightarrow$R)
CLM training
Generation tasks

GPT, LLaMA, Mistral

Encoder–Decoder

Bidirectional enc. $+$
autoregressive dec.
Seq-to-seq tasks

T5, BART, mBART

	Enc-Only	**Dec-Only**	**Enc-Dec**
Direction	Bidirectional	Left-to-right	Both
Best for	Classification, NER	Text generation	Translation, summary
Today's trend	Less common	**Dominant**	Niche

The trend since GPT-3 (2020): **decoder-only** models at scale can do almost everything, including understanding tasks.

How Transformers learn

Causal LM (decoder)

"The cat sat on the ______"

↓

Transformer

↓

softmax over vocab

$P(\text{"mat"}) = 0.42$

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(x_t \mid x_{<t})$$

Cross-entropy on **every** position

Teacher forcing: use true tokens as input, not model's predictions

Masked LM (encoder)

"The **[MASK]** sat on the mat"

↓

Transformer

↓

softmax over vocab

$P(\text{"cat"}) = 0.67$

$$\mathcal{L} = -\sum_{i \in \mathcal{M}} \log P(x_i \mid x_{\setminus \mathcal{M}})$$

Loss only on **masked** positions

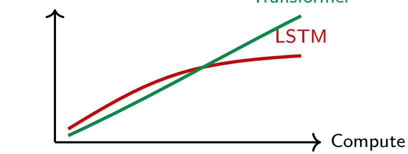
15% of tokens masked: 80% [MASK], 10% random, 10% unchanged

Since GPT-3: **causal LM is the dominant paradigm** — one training objective, scales to any task via prompting

Why Transformers won

	LSTM	Transformer
Parallelizable	No (h_t needs h_{t-1})	Yes (all tokens at once)
Long-range path	$O(n)$ steps	$O(1)$ (direct attention)
Context	Limited by hidden size	Full context window
Scaling	Diminishing returns	Log-linear improvement
Training speed	Slow (sequential)	Fast (GPU-friendly)

Scaling behavior



The scaling insight:

Transformers reliably improve with more data, parameters, and compute

This enabled the LLM revolution:
GPT-3, PaLM, LLaMA, Claude, ...

Computational complexity

Self-Attention

$O(n^2 \cdot d)$ time

$O(n^2)$ memory

($QK^\top$ is an $n \times n$ matrix)

Feed-Forward

$O(n \cdot d^2)$ time

$O(d^2)$ memory

(two $d \times 4d$ weight matrices)

Which dominates?

Short context ($n < d$)

FFN dominates

E.g., n=512, d=4096

Attention: $0.5\text{M} \cdot d$ vs FFN: $512 \cdot 16\text{M}$

Long context ($n > d$)

Attention dominates

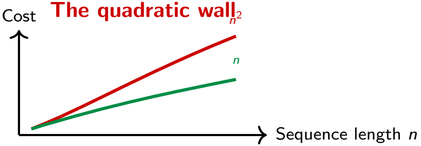
E.g., n=128K, d=4096

Doubling sequence length:

- Attention: $4\times$ more compute
- KV cache memory: $2\times$ more
- Attention matrix: $4\times$ more memory

128K tokens $\rightarrow$ 16 **billion** entries per head per layer

The quadratic wall



This quadratic bottleneck motivates FlashAttention, sparse patterns, and state space models

Transformer dimensions in practice

Model	Layers	d_{model}	Heads	d_{ff}	Params
BERT-base	12	768	12	3072	110M
BERT-large	24	1024	16	4096	340M
GPT-2	12	768	12	3072	117M
GPT-3	96	12288	96	49152	175B
LLaMA-2 7B	32	4096	32	11008	7B
LLaMA-2 70B	80	8192	64	28672	70B

$$d_k = \frac{d_{\text{model}}}{h}$$

Head dimension

$$d_{\text{ff}} \approx 4 \times d_{\text{model}}$$

FFN hidden size

$$\text{Params} \approx 12 \cdot N \cdot d^2$$

Rough estimate

Further reading

Attention & Transformers

- Vaswani et al. (2017), "Attention Is All You Need" — the original Transformer paper
- Bahdanau et al. (2015), "Neural Machine Translation by Jointly Learning to Align and Translate"
- Jay Alammar, "The Illustrated Transformer" —

Positional Encodings

- Su et al. (2021), "RoFormer: Enhanced Transformer with Rotary Position Embedding" (RoPE)

Architecture Variants & Surveys

- Devlin et al. (2019), "BERT: Pre-training of Deep Bidirectional Transformers" (encoder-only)
- Radford et al. (2018/2019), "Improving/Language Models are Unsupervised Multitask Learners" (GPT/GPT-2)
- Lin et al. (2022), "A Survey of Transformers" — comprehensive taxonomy

Questions?

Next: Tokenization