上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 22: Transformers

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

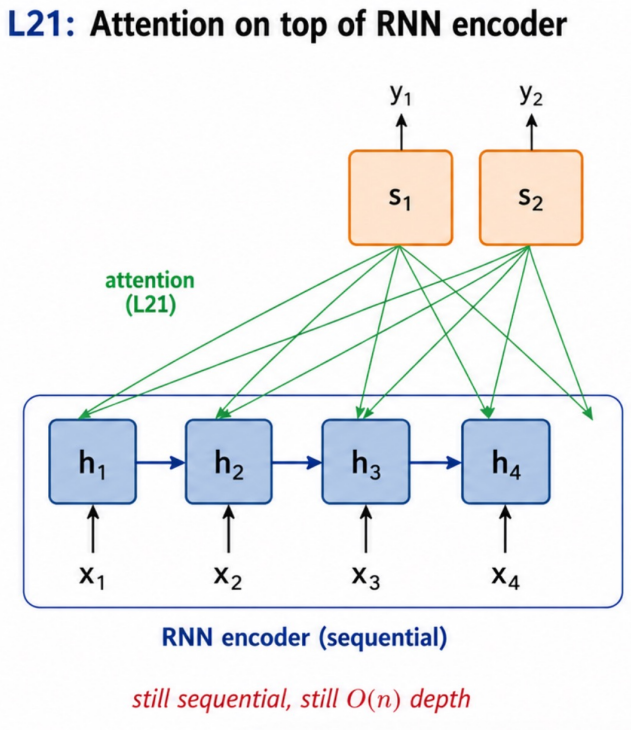
AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

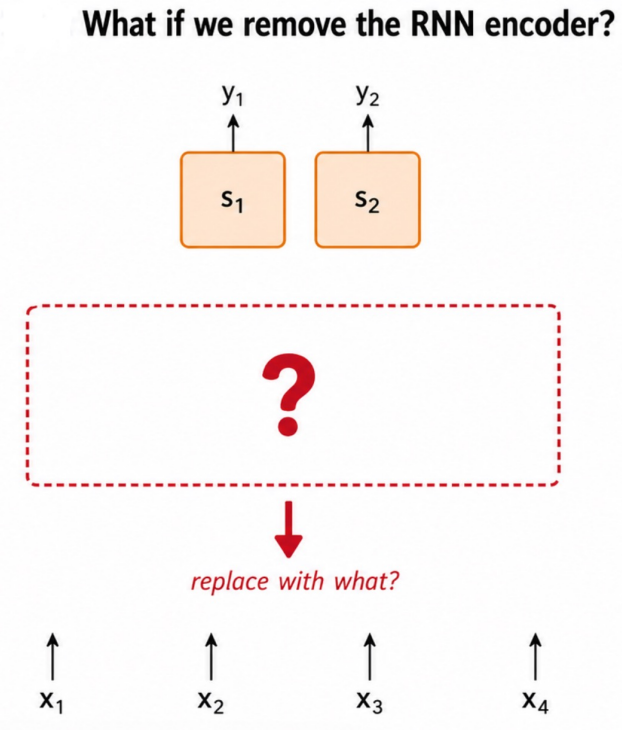
Bridge-in: Where Attention Left Us

L21 gave the decoder soft, content-based access to the encoder's full sequence — bottleneck broken.

Remaining issues:

- The encoder is still an RNN. Sequential. Slow. Gradient issue at long range.
- Attention solved the interface, not the backbone.





Question & Objectives

Question:

- What if attention isn't a helper on top of recurrence — but the only mixing operator? Can we kill the RNN entirely?

Objectives:

- **Analyze:** the $O(n^2 \cdot d)$ vs. $O(n \cdot d^2)$ tradeoff and predict when each wins
- **Evaluate:** why each Transformer component is necessary — what fails if you remove it
- **Create:** the right attention mask for a given task (encode / generate / transduce) from first principles

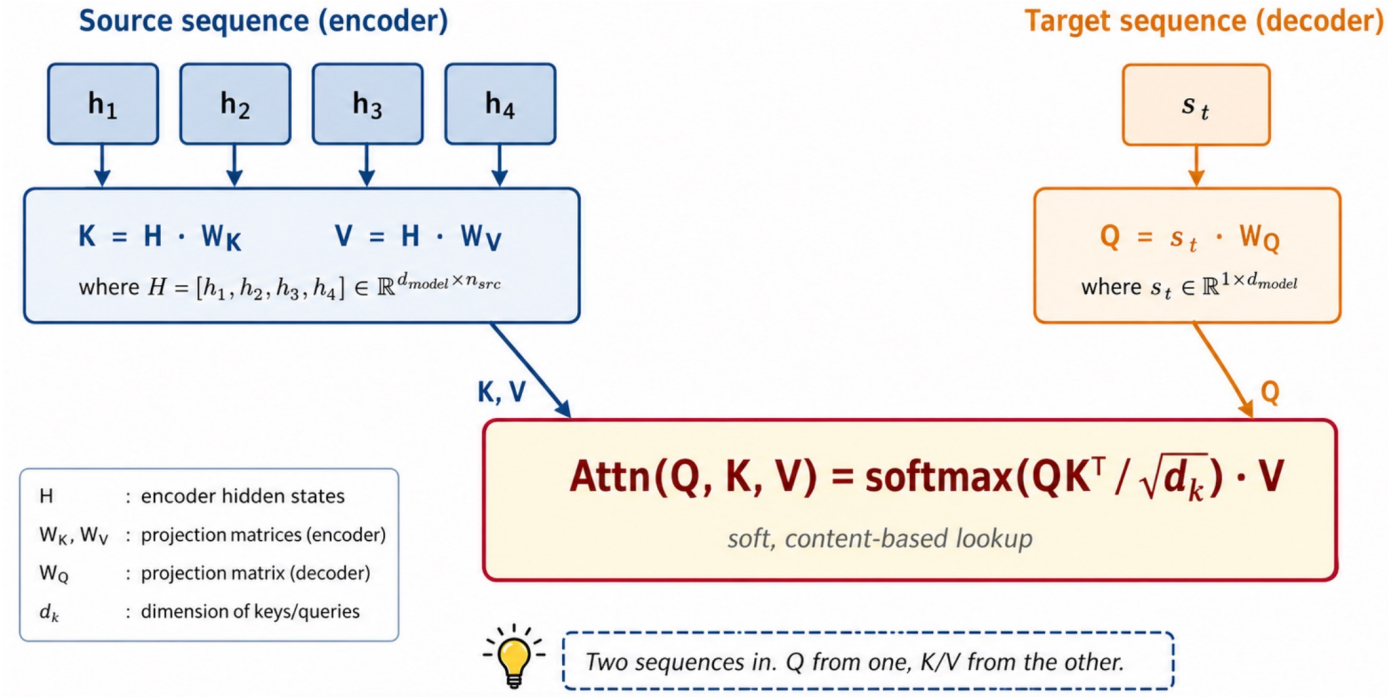
1. Self-Attention

Recall: Cross-Attention from L21

From L21: the decoder query attends over encoder keys & values.

- Q from decoder hidden state; K, V from encoder hidden states
- $\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$

Two sequences in, one sequence out. Q comes from one, K/V from the other.

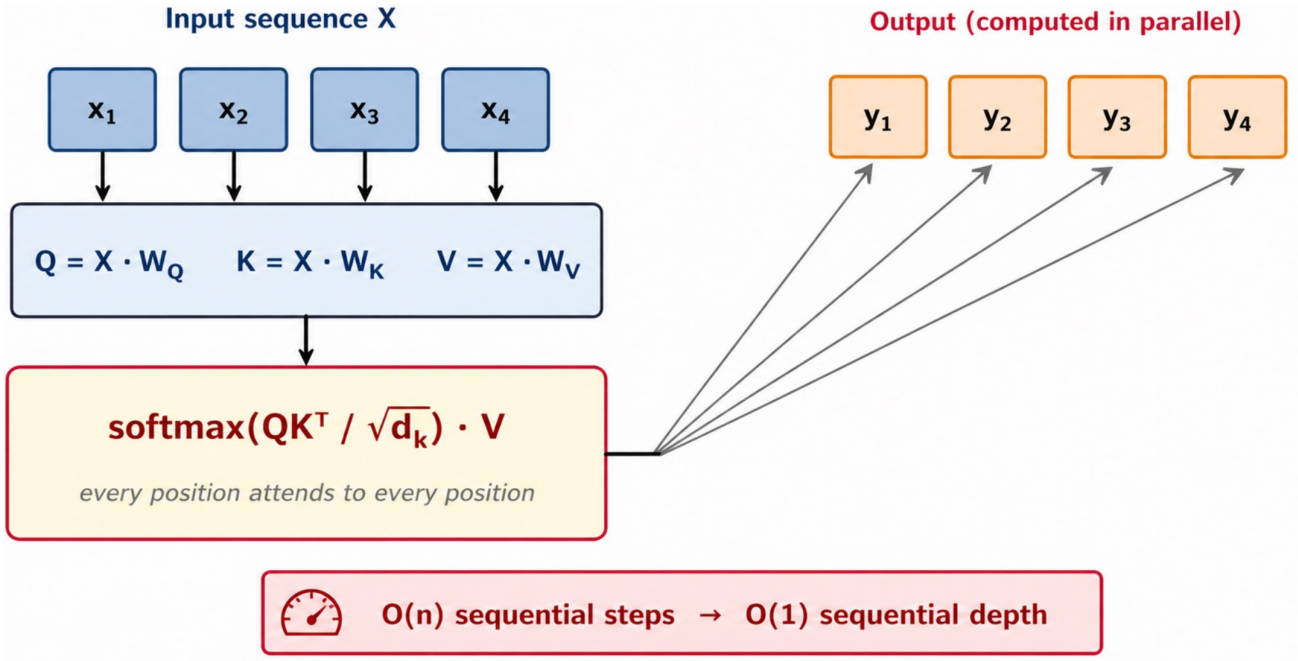


Self-Attention

What if Q, K, V all come from the same sequence?

- Each token computes its own query, key, value — and attends over all tokens (including itself).
- $Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad \text{where } X \in \mathbb{R}^{n \times d}$

No recurrence anywhere. Every output position is computed from every input position in parallel.

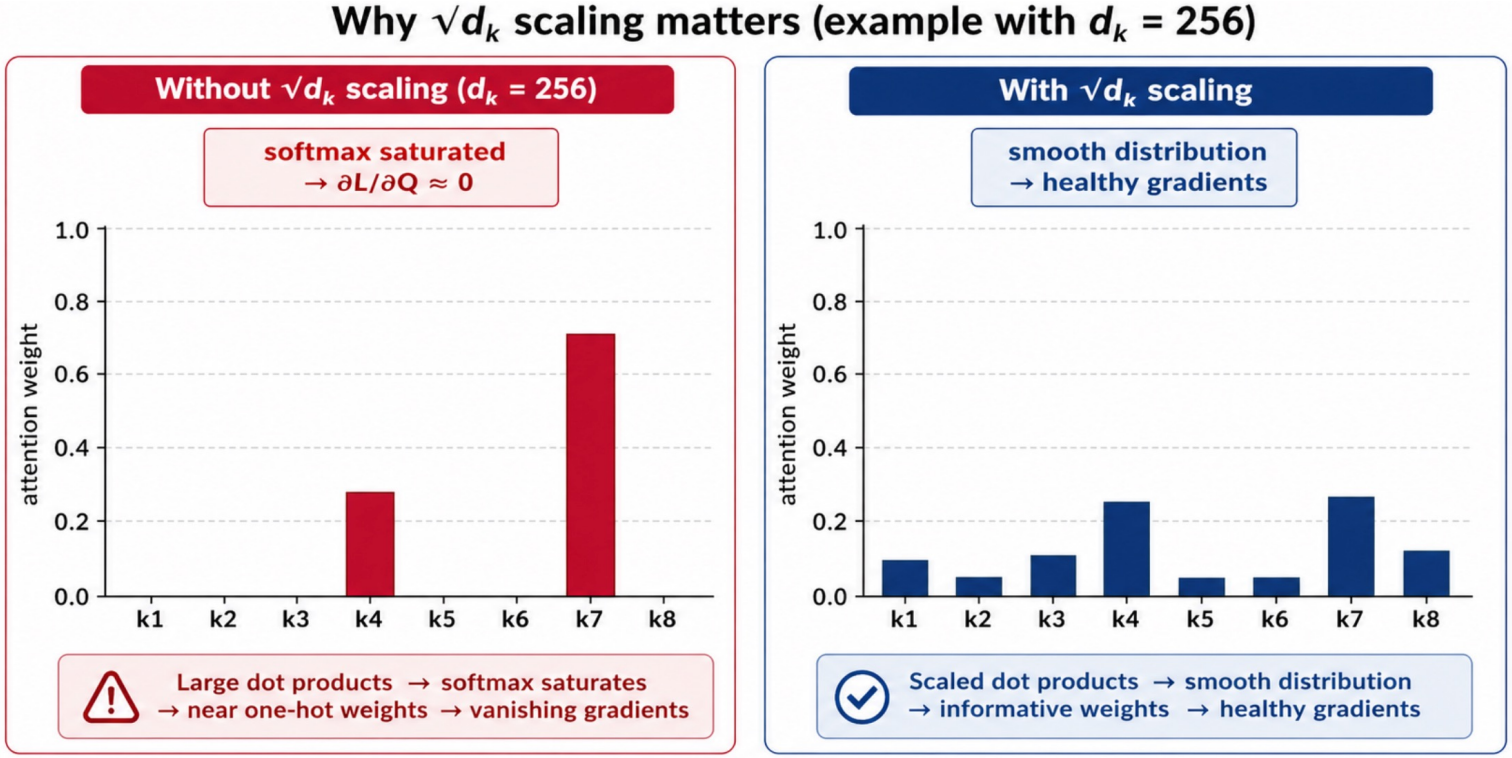


Scaled Dot-Product: Why the $\sqrt{d_k}$?

Without scaling: as d_k grows, QK^T entries grow with std $\propto \sqrt{d\_k}$ → softmax saturates → gradients vanish.

The fix: divide by $\sqrt{d_k}$ to keep variance of pre-softmax scores ≈ 1.

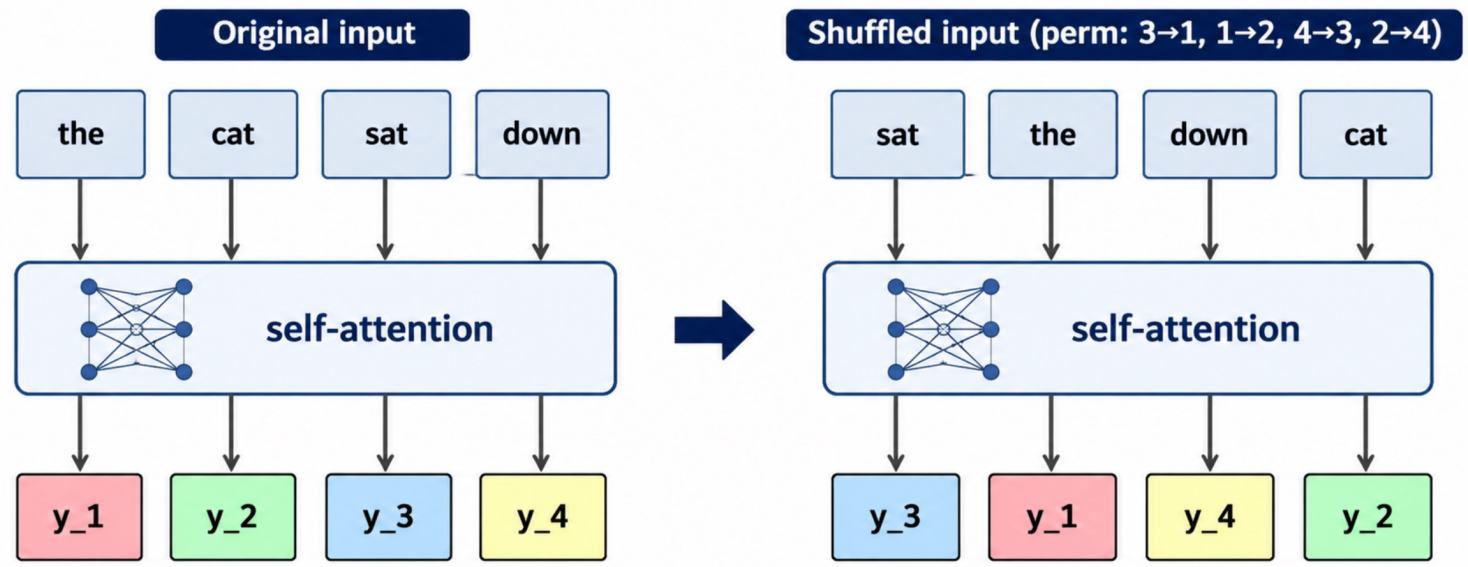
- If $q, k \in \mathbb{R}^{d_k}, \quad q_i, k_i \sim \mathcal{N}(0,1)$, then $\mathrm{Var}(q \cdot k) = d_k$.



What Just Broke #1: Permutation Equivariance

Example: 4 tokens ["the", "cat", "sat", "down"]. Compute self-attention. Now shuffle the input. Recompute.

- Result: output is the **same** — just shuffled the same way.
- Self-attention treats the sequence as a **SET**, not a sequence.
- Discussion: Why is this catastrophic for language? *Give an example where word order changes meaning.*



2. Putting Order Back In

The Permutation Problem

Self-attention is permutation-equivariant. We need to inject position information somewhere.

Original Transformer: add a position vector $p_t \in \mathbb{R}^d$ to each input embedding e_t.

- $x_t = e_t + p_t$

Two choices for p_t:

- Learned (lookup table indexed by position)
- Fixed (a function of t)

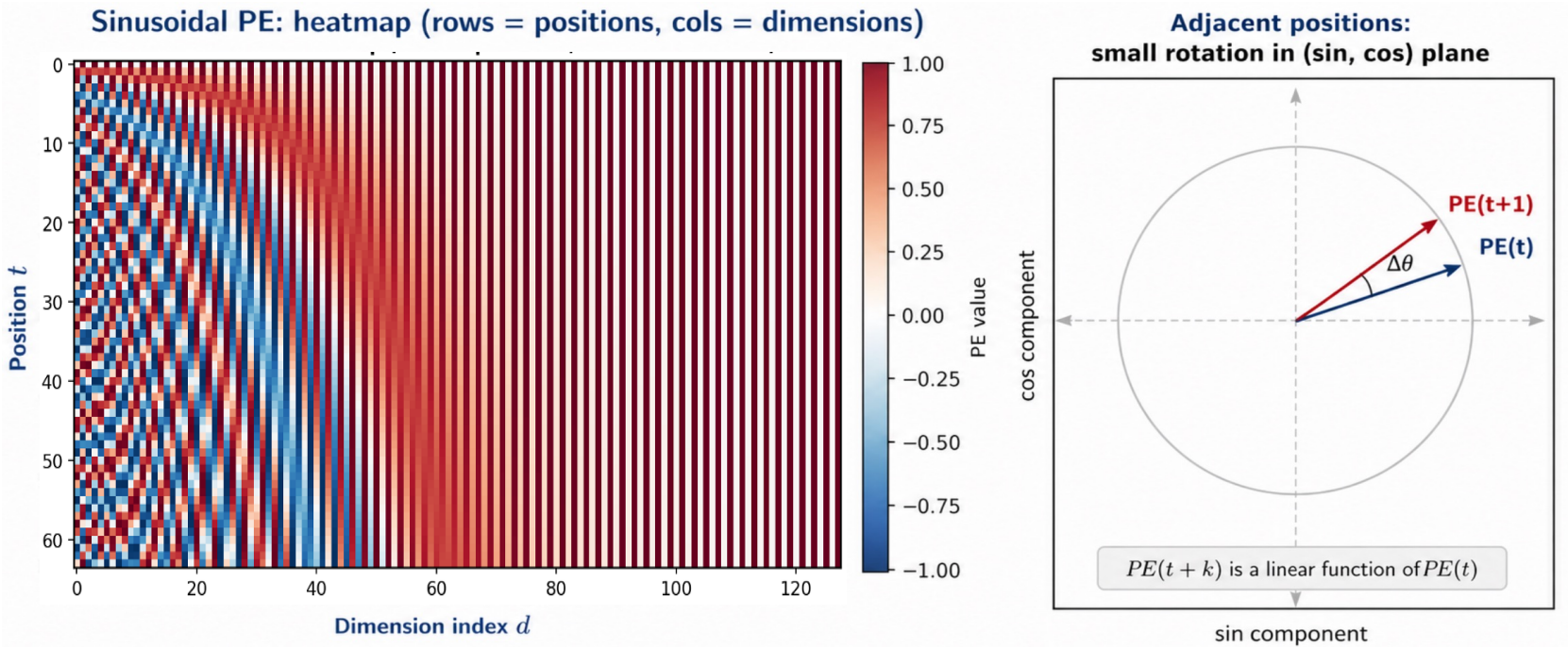
Sinusoidal Positional Encoding

Original Transformer's choice — fixed, not learned:

$$\mathrm{PE}(t, 2i) = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \quad \mathrm{PE}(t, 2i+1) = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right)$$

Why this form? Two reasons:

1. **Extrapolation:** can encode positions longer than seen in training.
2. **Linear-relative encoding:** PE(t+k) is a linear function of PE(t) — model can learn relative positions easily.



Sinusoidal vs. Learned: Why Choose Either?

Learned PE (used by BERT, GPT-2):

- Pro: more expressive, no inductive bias to be wrong
- Con: hard cap on sequence length (one row per position); doesn't extrapolate

Sinusoidal PE (original Transformer):

- Pro: extrapolates to unseen lengths; deterministic
- Con: empirically often slightly weaker than learned at training length

Empirical finding (original paper): the two perform within noise on translation. Learned won the early LLM era; the long-context era brought sinusoidal-style ideas back.

What Replaced Both: RoPE & ALiBi

The 2026 reality: essentially no frontier LLM uses original sinusoidal or learned absolute PE.

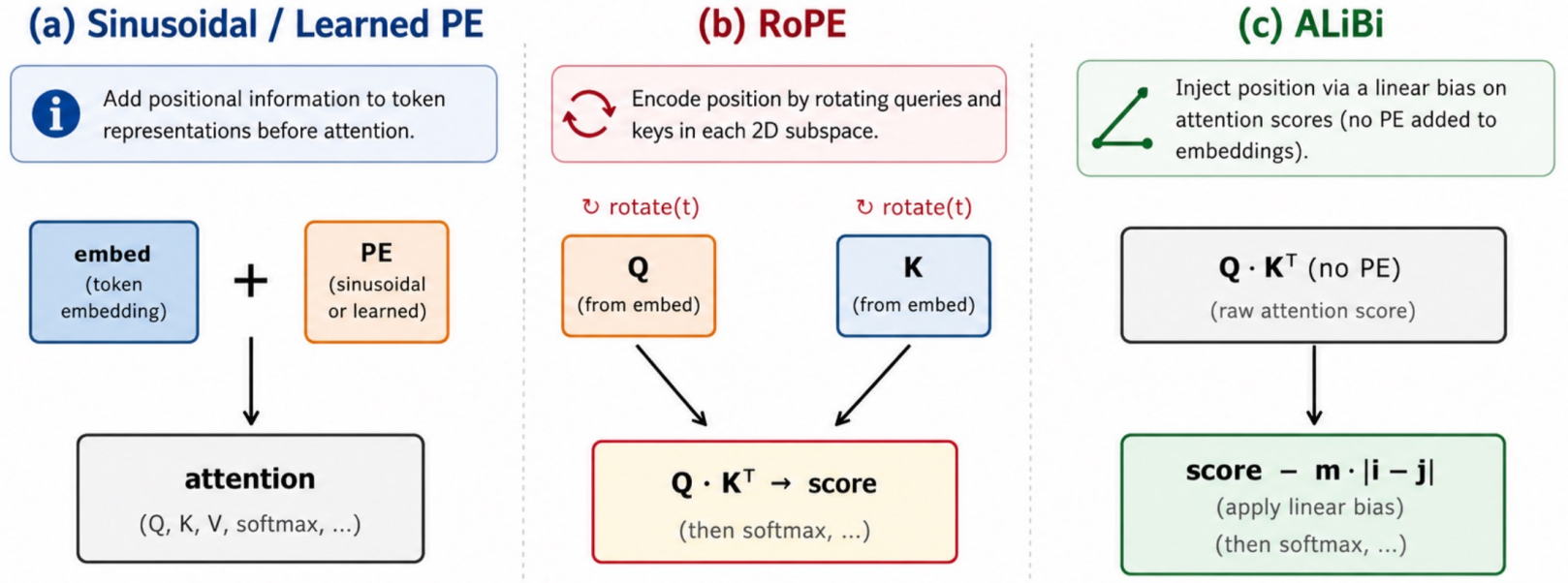
RoPE (Rotary Position Embedding, Su et al. 2021):

- Rotate Q and K by an angle proportional to position before dot product. Encodes relative position directly into attention scores.

ALiBi (Press et al. 2022):

- Add a linear bias $-m \cdot |i - j|$ to attention scores. No learned PE at all.

Both exploit the same insight: position should affect the attention score, not the token's value.



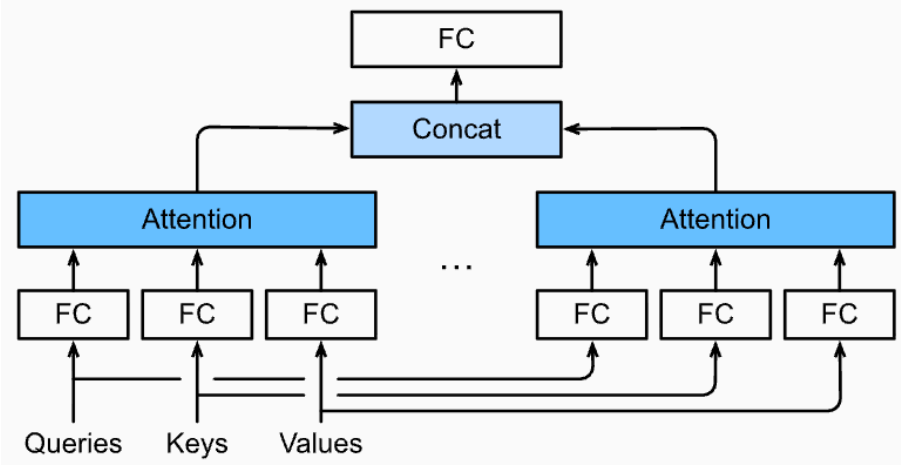
3. One Head is Not Enough

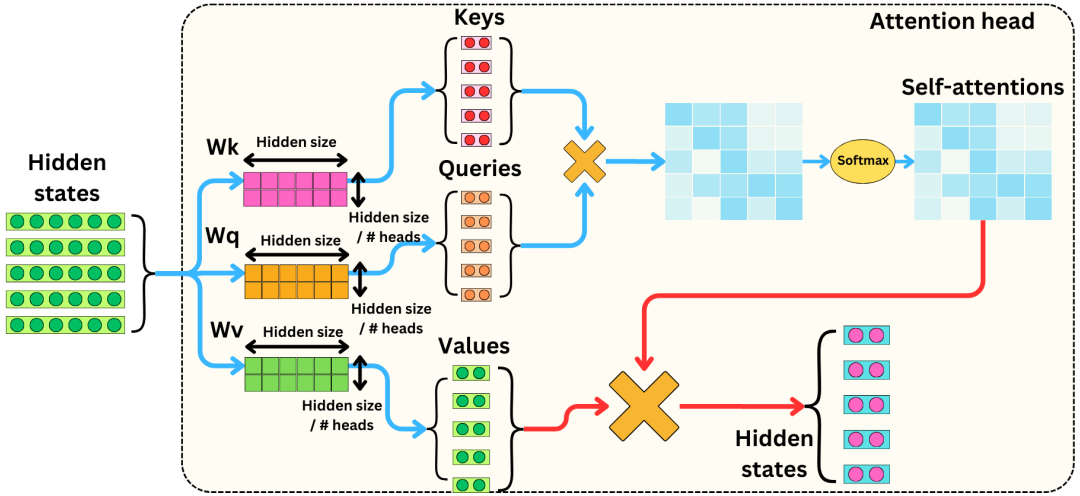
The Capacity Argument for Multiple Heads

A single self-attention layer produces **ONE** softmax map per token: one relational pattern.

But a token in language has many simultaneous relations: subject-verb, modifier, position, semantics…

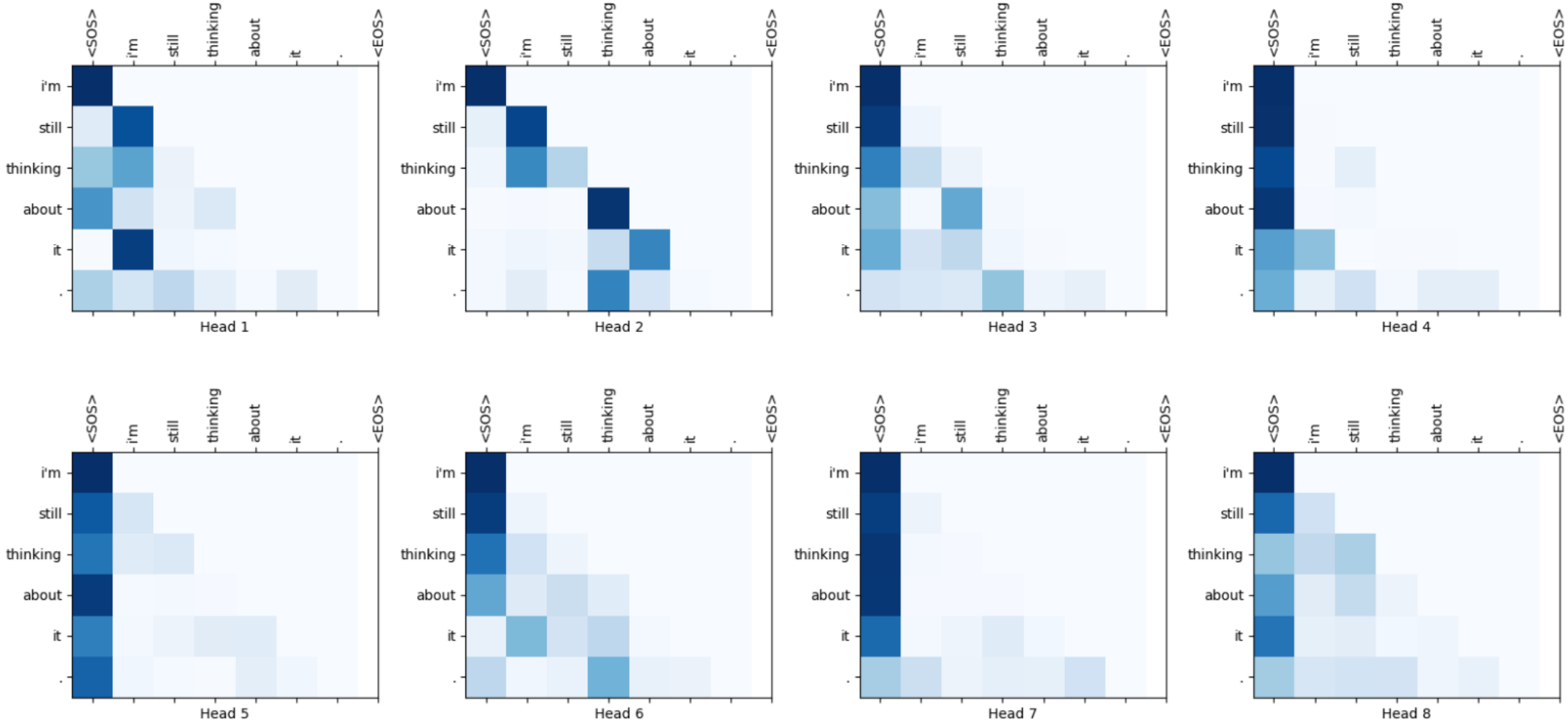
- One softmax can only "look at" one thing at a time. Averaging all relations into one map loses them.
- The fix: split each token into h parts => run h attentions in parallel.





What Heads Actually Learn

Empirical observation: trained heads specialize.

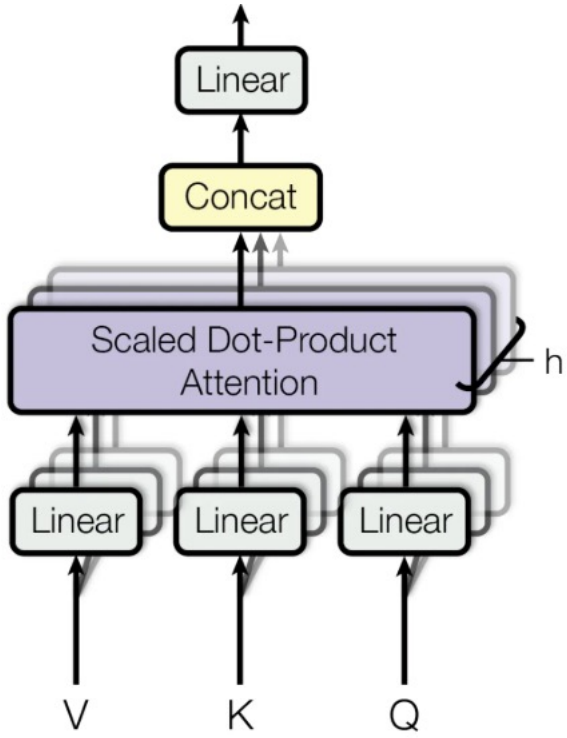


Multi-Head: The Mechanics

Full pipeline for one layer:

1. Project X to h sets of (Q_i, K_i, V_i), each of dim $\frac{d}{h}$
2. Compute h scaled dot-product attentions in parallel
3. Concatenate to get back to dim d
4. Apply output projection W_O

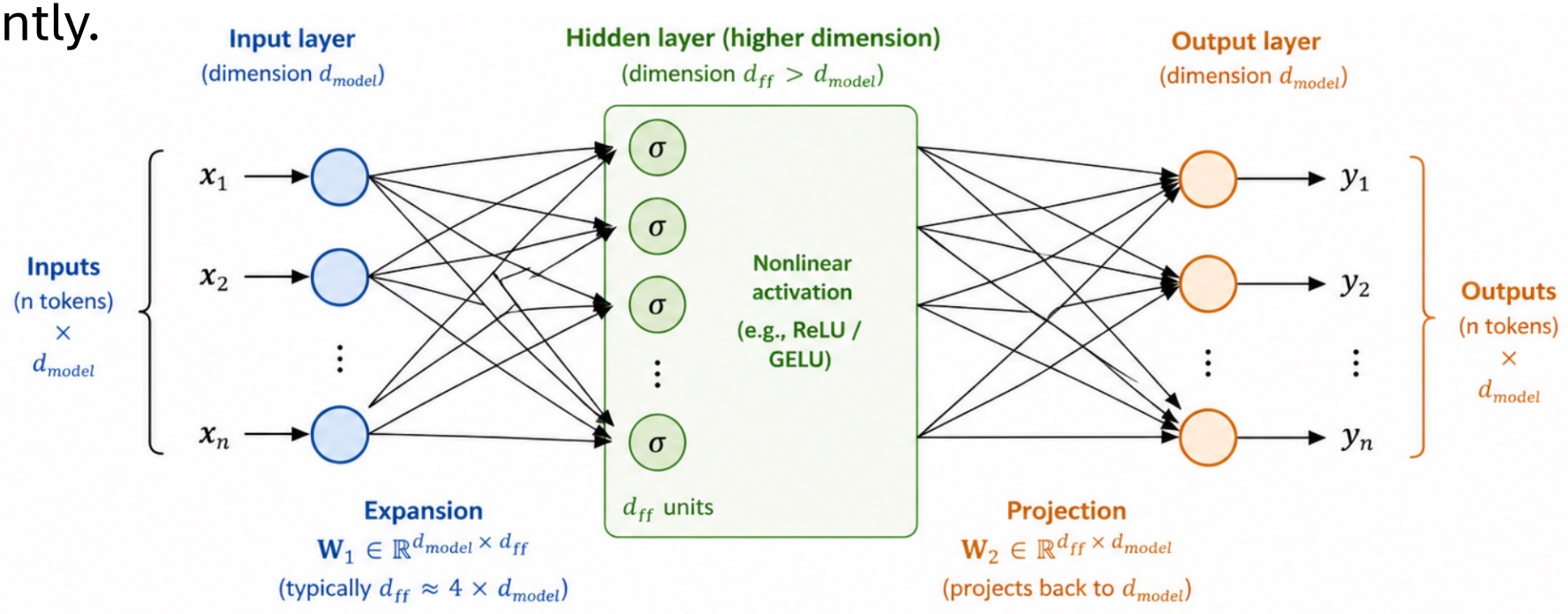
- Typical: d = 512, h = 8, so $d_k = d_v = 64$ per head.



4. Building the Transformer Network

What Self-Attention Doesn't Do: Per-Token Computation

- Self-attention is a **MIXING** operator: each output is a weighted average of value vectors.
- What's missing: a place for each token to **THINK** about its own (now contextualized) representation, nonlinearly.
- The fix: a position-wise **feed-forward network (FFN)**, applied to each token independently.



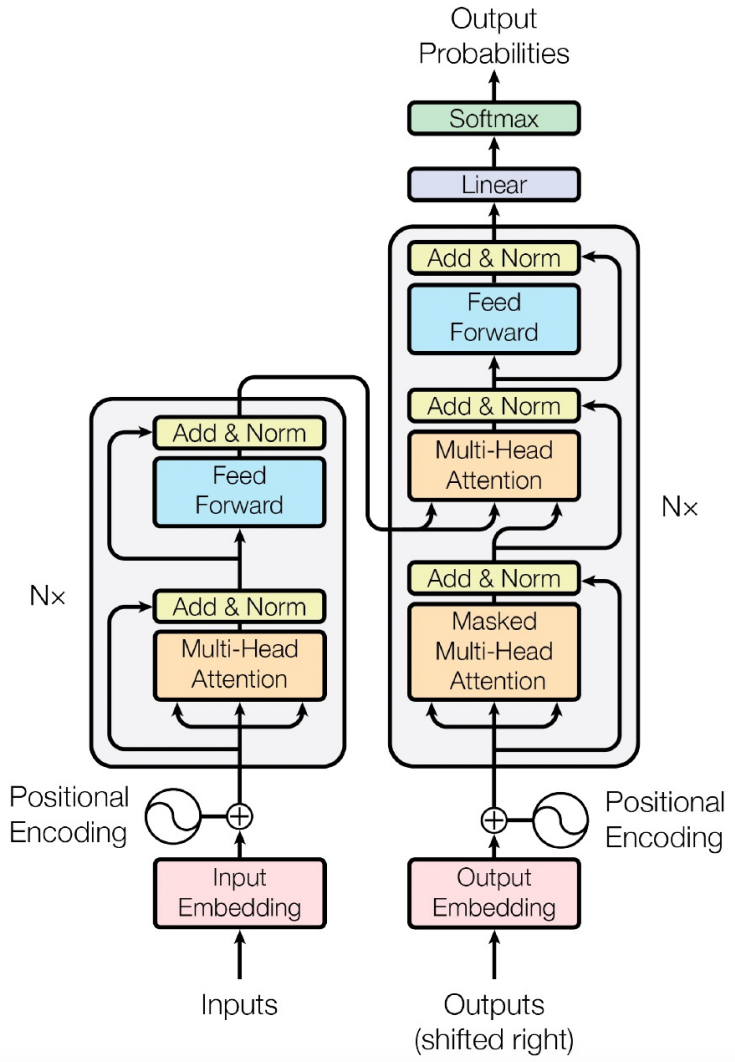
Modules for Training Stability

Two causes of instability:

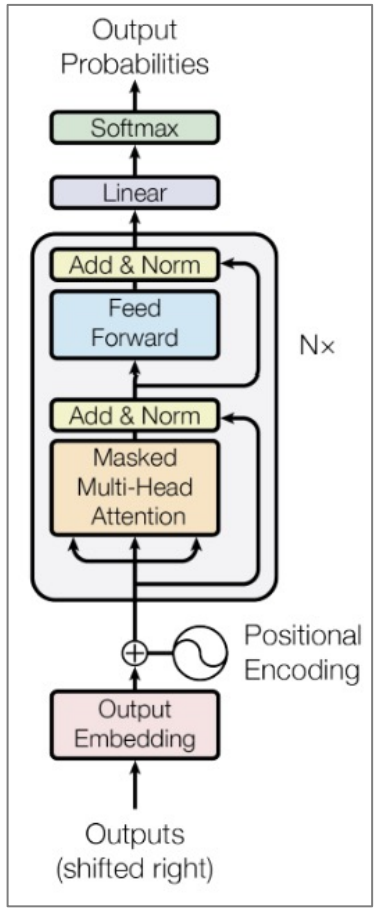
- Gradient flow: deep stacks of attention + FFN have the same vanishing/exploding issue as deep nets generally.
- Activation drift: the scale of activations changes layer-to-layer, destabilizing the softmax.

Two off-the-shelf fixes from CV:

- Residual connections (He et al. 2015, ResNet)
- Layer normalization (Ba et al. 2016)



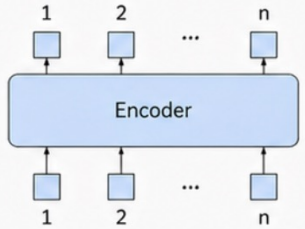
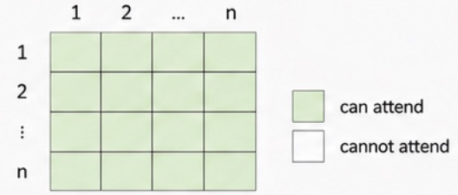
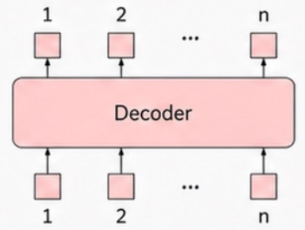
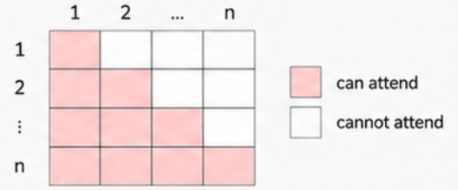
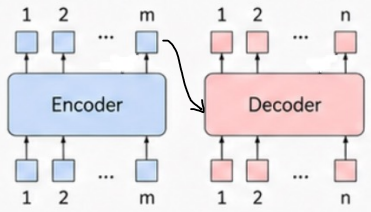
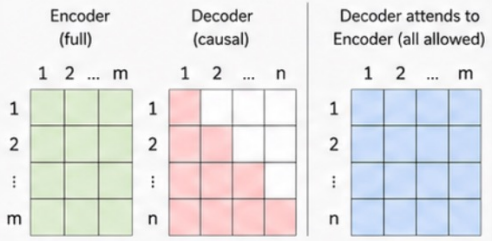
Encoder-Decoder



Decoder-only

Three Types of Transformer Architectures

- All three use the same Transformer block. The differences:

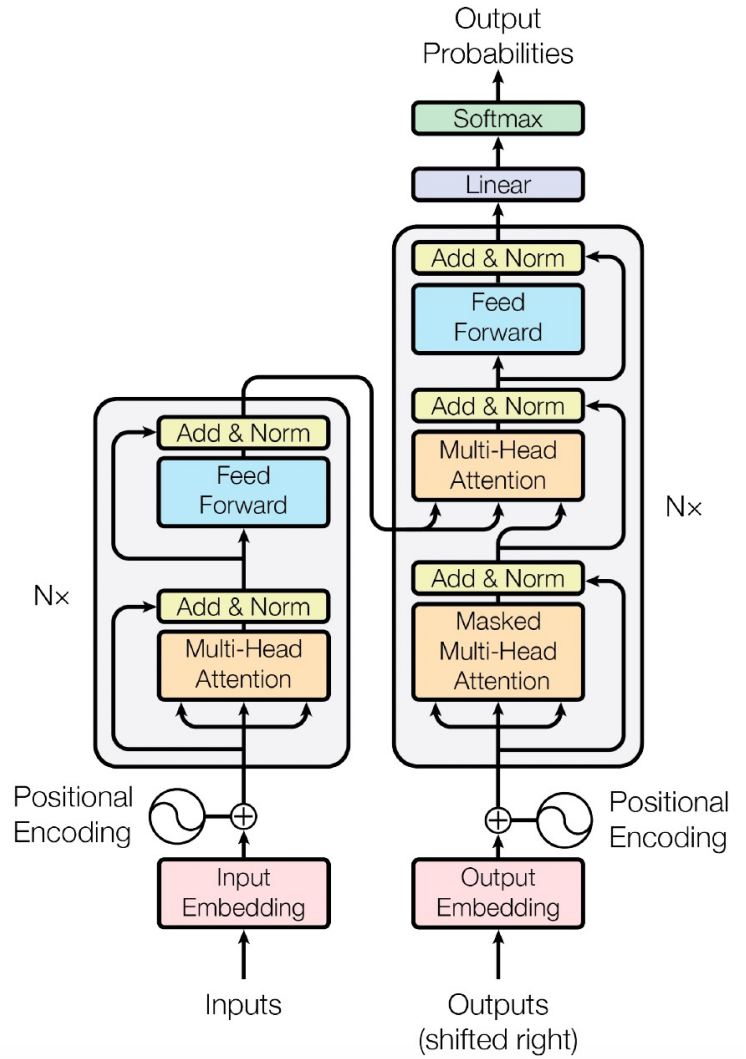
Variant	Mask	Cross-attn?	Example	Best for
Encoder-only 	**None** (full bidirectional) 	No	BERT	Classification, embeddings
Decoder-only 	**Causal** (lower-triangular) 	No	GPT	Generation, completion
Encoder-decoder 	(Self-attention masks) 	Yes	T5, original Transformer	Translation, summarization

Encoder-Decoder

Inside the decoder of an encoder-decoder Transformer:

1. Masked self-attention over the partially-generated output (causal)
2. Cross-attention with Q from decoder, K/V from encoder output

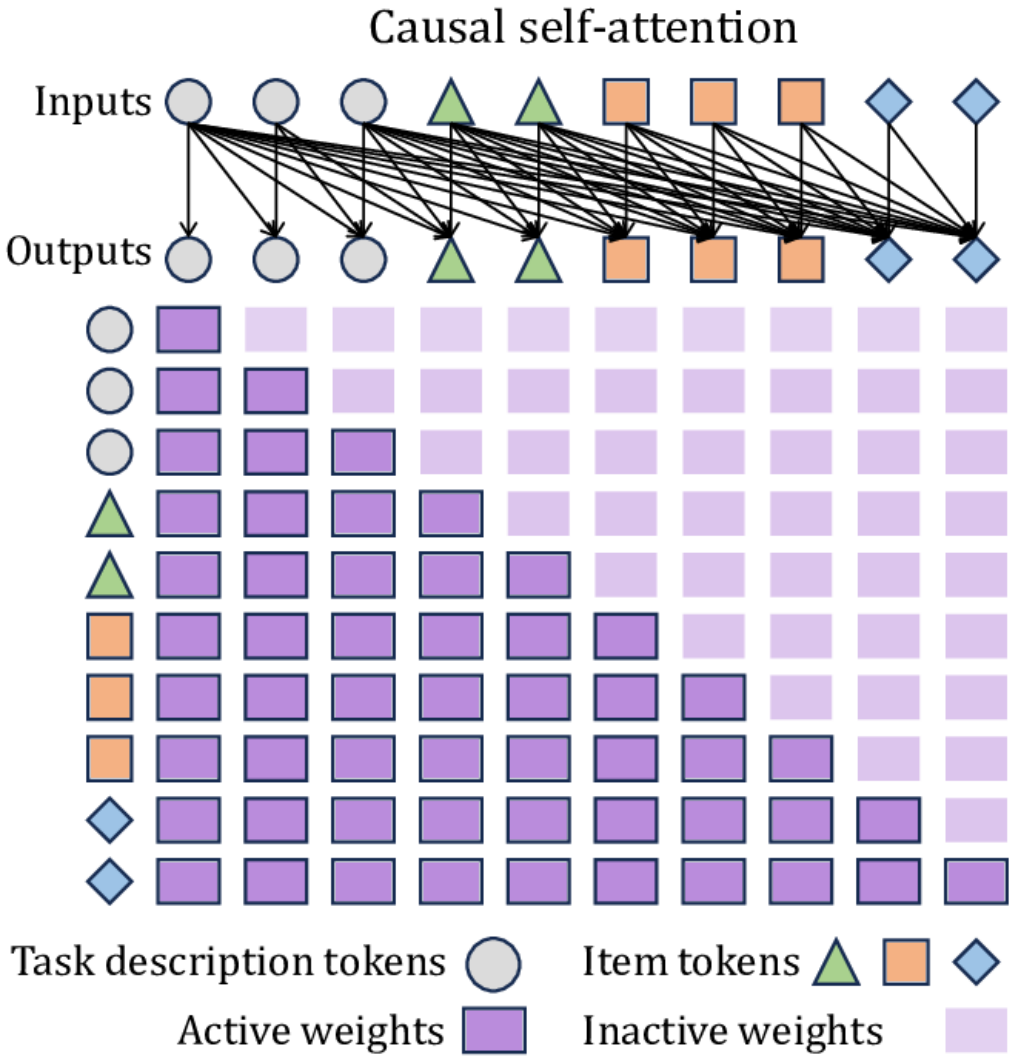
The original 2017 Transformer is this.



Encoder-Decoder

Causal Masking: Generation from First Principles

- Why generation needs masking: at training time we predict token t+1 from tokens 1..t in PARALLEL, not one at a time.
- Trick: feed the whole sequence in, but BLOCK every token from attending to future positions.
- Implementation: add $-\infty$ to disallowed entries of QK^T before softmax → those weights become 0.



What Transformer Brings

Parallel training:

- Every position computed simultaneously per layer → modern GPUs saturate; training time drops by orders of magnitude.

Constant gradient path:

- Any-to-any in O(1) layers → long-range dependencies trained directly, not through recurrent steps.

Scale unlocks:

- The architecture scales smoothly with parameters, data, and compute — this is what made GPT-2/3/4 possible.

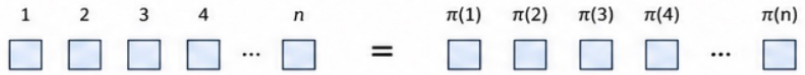
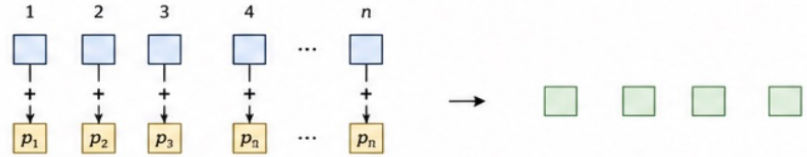
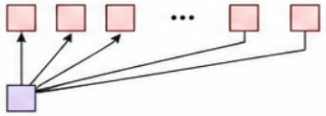
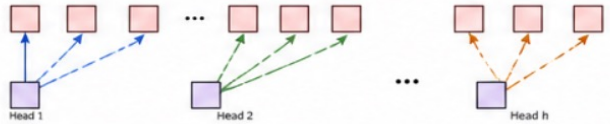
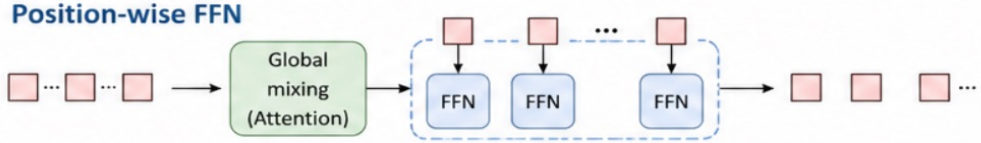
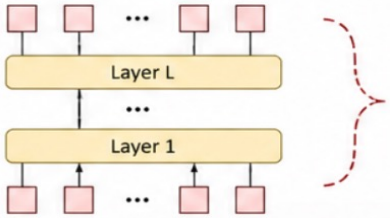
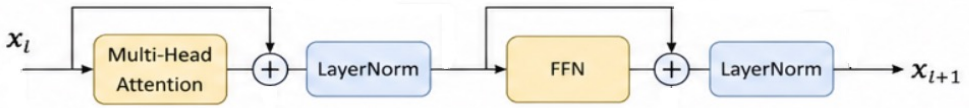
What It Costs

- Quadratic compute & memory: $O(n^2)$ attention; KV cache scales as O(n) per layer per head; long context is expensive.
- Data hunger: weak inductive bias → needs orders of magnitude more data than CNNs/RNNs.
- No native streaming: every token attends to the whole past — there's no compact recurrent state to update.

When an RNN or state-space model still beats a Transformer in 2026 (see extension this week):

- Streaming inference at low latency (audio, sensor signals)
- Edge / on-device inference where KV cache memory dominates
- Extremely long sequences (>100k tokens) where $O(n^2)$ is prohibitive

Summary

What broke	The fix
Permutation equivariance  Model output is unchanged under any permutation π of inputs.	**Positional encoding (sinusoidal / learned / RoPE / ALiBi)**  Inject position information so order matters.
Single relational pattern per token  Each token attends with one pattern (one set of Q,K,V).	**Multi-head attention**  Multiple heads learn diverse relational patterns.
No per-token computation  Only global mixing (e.g., attention); no independent per-token transformation.	**Position-wise FFN**  Apply the same FFN to each position independently.
Training instability at depth  Gradients must pass through many layers directly → vanishing / exploding.	**Residual + LayerNorm (pre-LN)**  Skip connections preserve gradient flow; pre-LN stabilizes training.