

COMS4995W32

Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Fall 2025



Transformer++



Recap on Transformer

Input:

the cat sat on

Step 1:

Embeddings ($n \times d$)

Step 2:

Q

K

V

Step 3:

Scores = $Q @ K^T$

Step 4:

Causal Mask

Step 5:

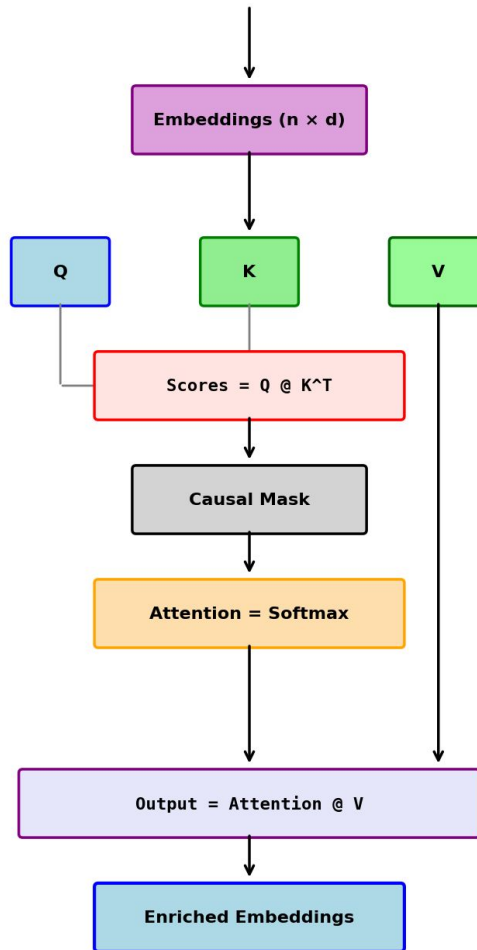
Attention = Softmax

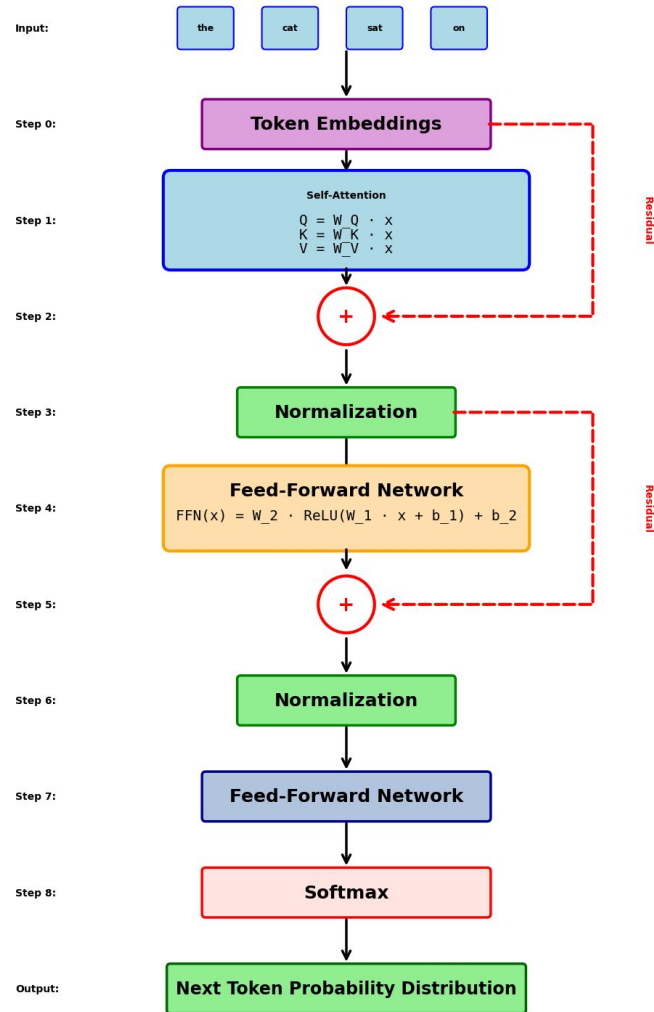
Step 6:

Output = Attention @ V

Output:

Enriched Embeddings





Attention is All You Need

Agenda++



- Positional Encoding & Masking
- Multi-head Self-Attention
- Normalization & Optimization
- Model Family
- Decoding Strategies
- Research Frontiers



Positional Encoding

Why Position Matters 🤔



Transformer treats all tokens in parallel → no notion of order

Think of it as giving every token a “**position tag**”

- Just like labeling words 1, 2, 3, ... in a sentence

2 main strategies:

- **Absolute**: encode position directly (e.g., [sinusoidal](#))
- **Relative**: express distance between tokens (e.g., [RoPE](#))

Classic Sinusoidal Encoding



Position embeddings are deterministic **sine** and **cosine** functions

For a token at position **t**:

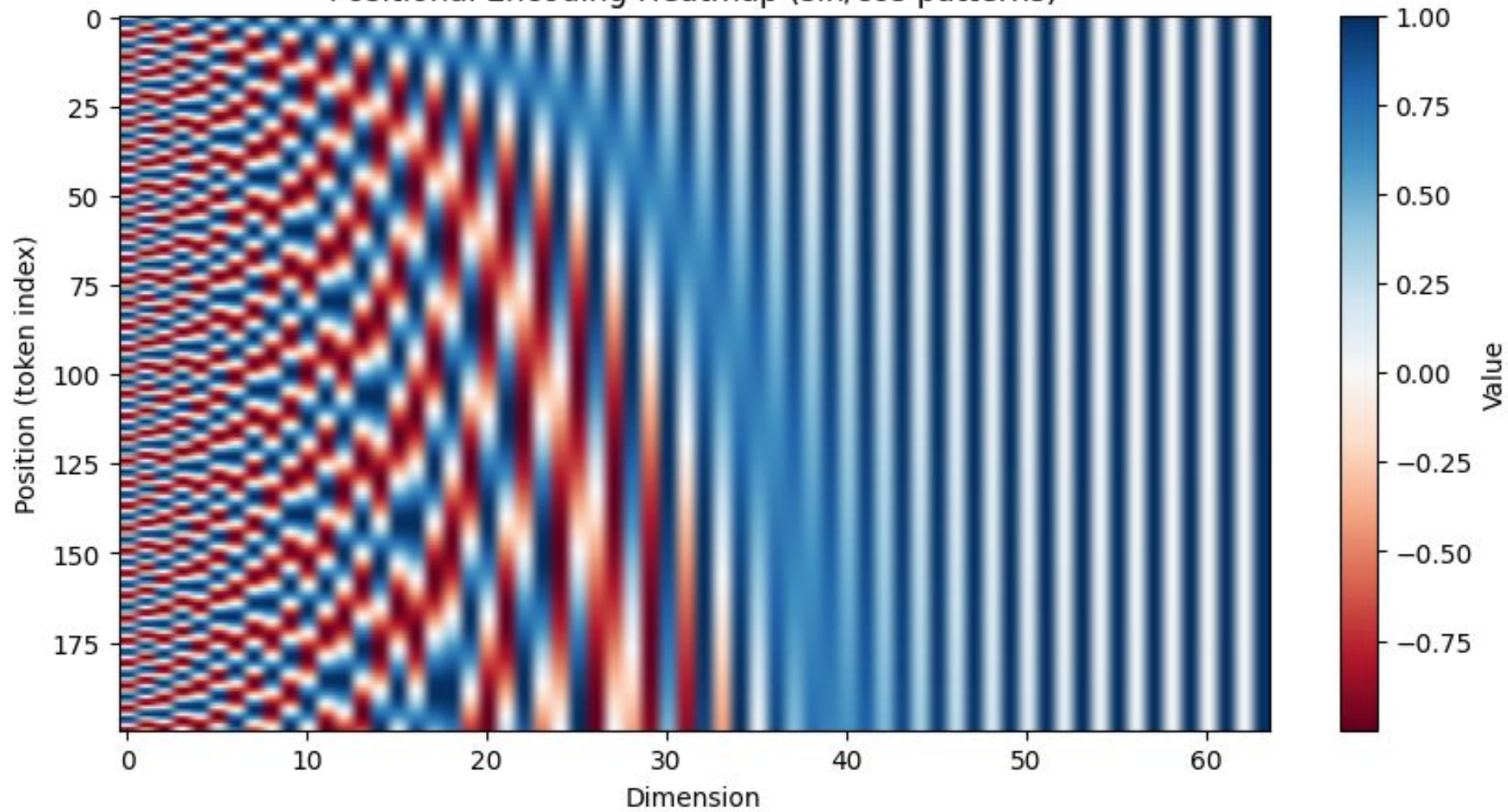
Even dimensions $\rightarrow PE(t, 2i) = \sin(t / (10000 ^ (2i / d_model)))$

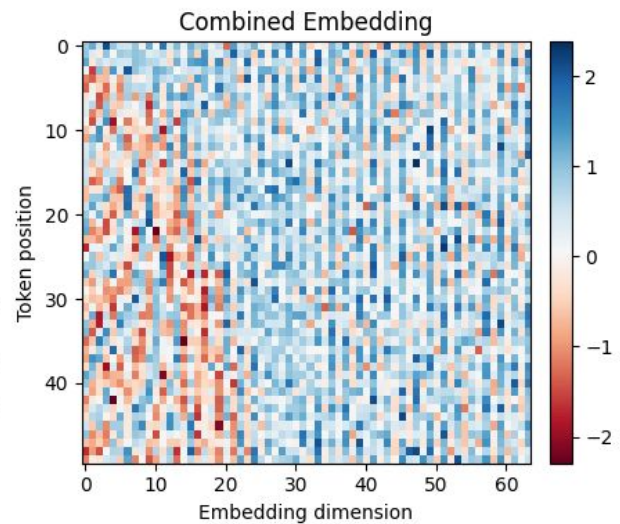
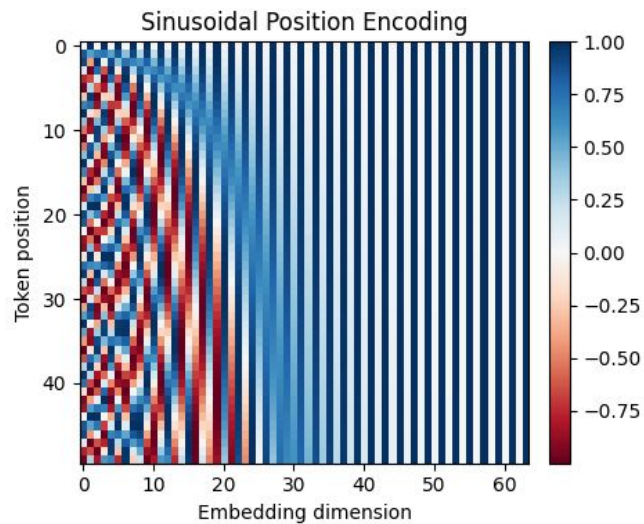
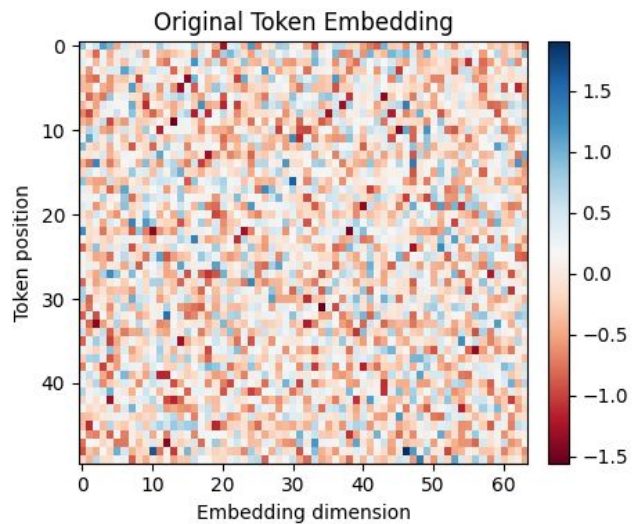
Odd dimensions $\rightarrow PE(t, 2i+1) = \cos(t / (10000 ^ (2i / d_model)))$

Continuous and smooth \rightarrow generalizes to unseen positions

Limitation: fixed frequency range, weaker for very long sequences

Positional Encoding Heatmap (sin/cos patterns)





Rotary Position Embedding (RoPE)



Idea: Instead of giving each token a fixed position number, **RoPE rotates** token representations in a smooth way across the sequence

How it works:

- Each token's query (Q) and key (K) vectors are placed on tiny 2-D circles
- As position increases, we rotate these vectors a little more each time
- Close tokens = small rotation difference → easy to attend
- Distant tokens = larger rotation difference → harder to attend

Used in LLaMA, ChatGPT, Gemini, and most modern LLMs

$$Attention\ Score_{1,3} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} & q_{15} & q_{16} \end{bmatrix} \cdot \begin{bmatrix} \cos 2\theta_a & -\sin 2\theta_a & 0 & 0 & 0 & 0 \\ \sin 2\theta_a & \cos 2\theta_a & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos 2\theta_b & -\sin 2\theta_b & 0 & 0 \\ 0 & 0 & \sin 2\theta_b & \cos 2\theta_b & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos 2\theta_c & -\sin 2\theta_c \\ 0 & 0 & 0 & 0 & \sin 2\theta_c & \cos 2\theta_c \end{bmatrix} \cdot \begin{bmatrix} k_{31} \\ k_{32} \\ k_{33} \\ k_{34} \\ k_{35} \\ k_{36} \end{bmatrix} =$$

$$= \begin{bmatrix} q_{11} & q_{12} \end{bmatrix} \cdot \begin{bmatrix} \cos 2\theta_a & -\sin 2\theta_a \\ \sin 2\theta_a & \cos 2\theta_a \end{bmatrix} \cdot \begin{bmatrix} k_{31} \\ k_{32} \end{bmatrix} + \begin{bmatrix} q_{13} & q_{14} \end{bmatrix} \cdot \begin{bmatrix} \cos 2\theta_b & -\sin 2\theta_b \\ \sin 2\theta_b & \cos 2\theta_b \end{bmatrix} \cdot \begin{bmatrix} k_{33} \\ k_{34} \end{bmatrix} + \begin{bmatrix} q_{15} & q_{16} \end{bmatrix} \cdot \begin{bmatrix} \cos 2\theta_c & -\sin 2\theta_c \\ \sin 2\theta_c & \cos 2\theta_c \end{bmatrix} \cdot \begin{bmatrix} k_{35} \\ k_{36} \end{bmatrix} = a_{13}$$



Multi-Head Attention

Motivation



A single attention head can only flow one type of information

Multi-head attention allows flowing various information in parallel

Intuitively:

→ Each head = one “expert” view of the data

→ Together = an ensemble of perspectives working in parallel

Stabilizes training through feature diversity

Mechanism



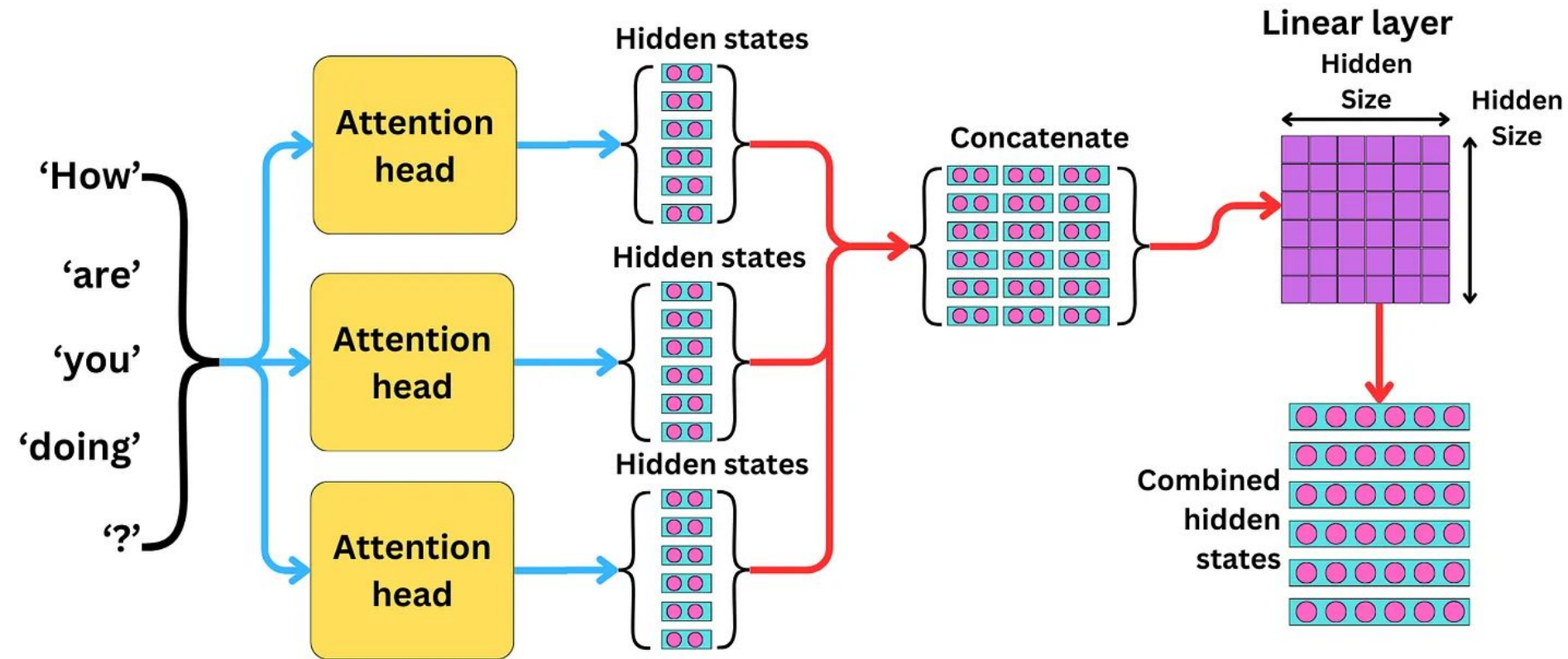
Split the hidden dimension (d_{model}) into h smaller parts (one per head):

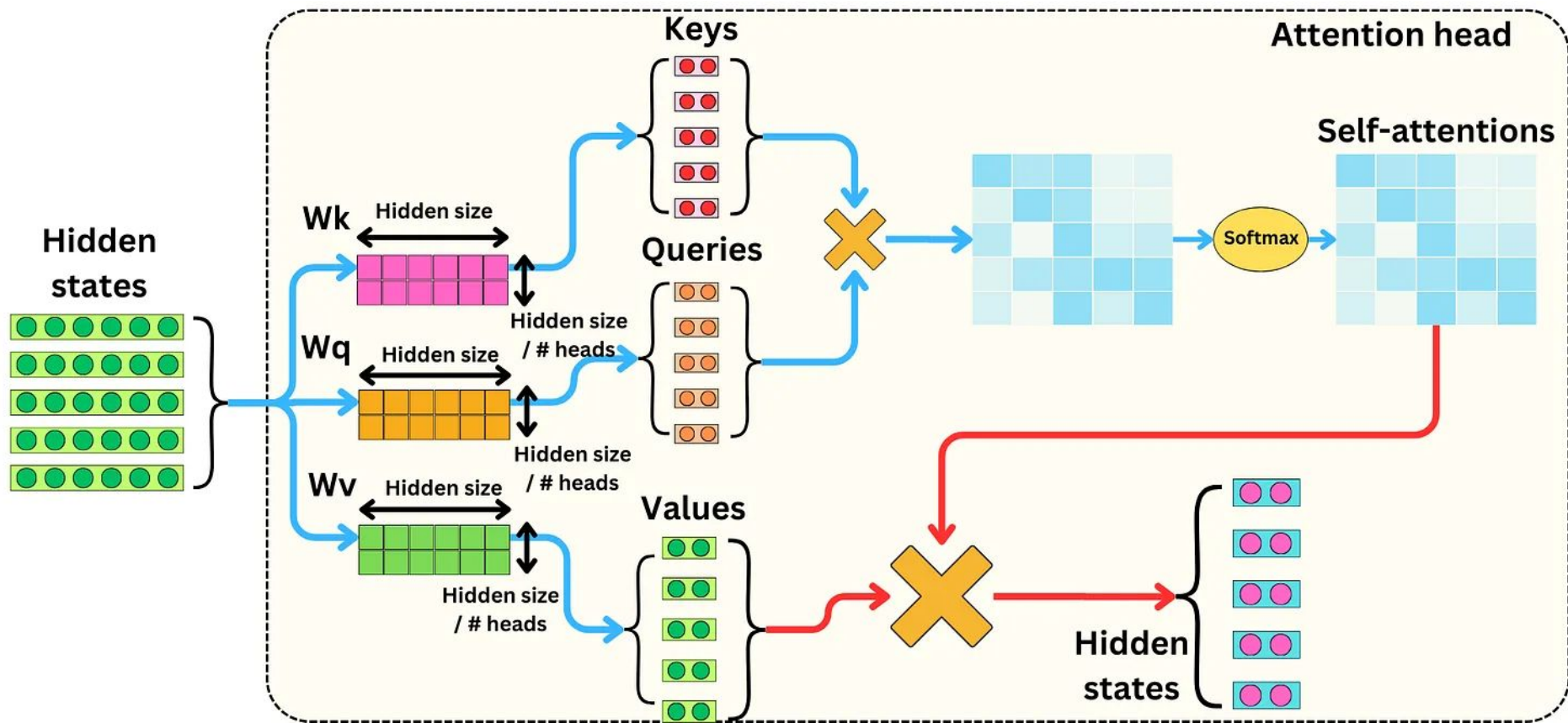
$$d_h = d_{\text{model}} // h$$

Each head performs self-attention independently:

$$\text{Attention}(Q, K, V) = \text{softmax}((QK^T) / \sqrt{d_h}) \times V$$

Concatenate all head outputs and project back to d_{model}







Normalization & Optimization

To Make Training Stable



Problem

Very deep networks \rightarrow gradients explode or vanish \rightarrow unstable training

Solution

Residual connections + **Normalization** keep signal flow smooth and balanced

[Residual connections] carries the original signal forward/backward

[Normalization] scales activations, so each layer receives **consistent** inputs



LayerNorm

Given a token's hidden vector $x = [x_1, x_2, \dots, x_d]$

Compute mean and variance:

$$\mu = \text{mean}(x)$$

$$\sigma^2 = \text{mean}((x - \mu)^2)$$

Normalize each feature:

$$\text{LN}(x) = \gamma * (x - \mu) / \text{sqrt}(\sigma^2 + \epsilon) + \beta$$

where γ and β are **learnable** scale and shift parameters



RMSNorm

Compute root mean square (RMS):

$$\text{rms}(x) = \sqrt{\text{mean}(x^2)}$$

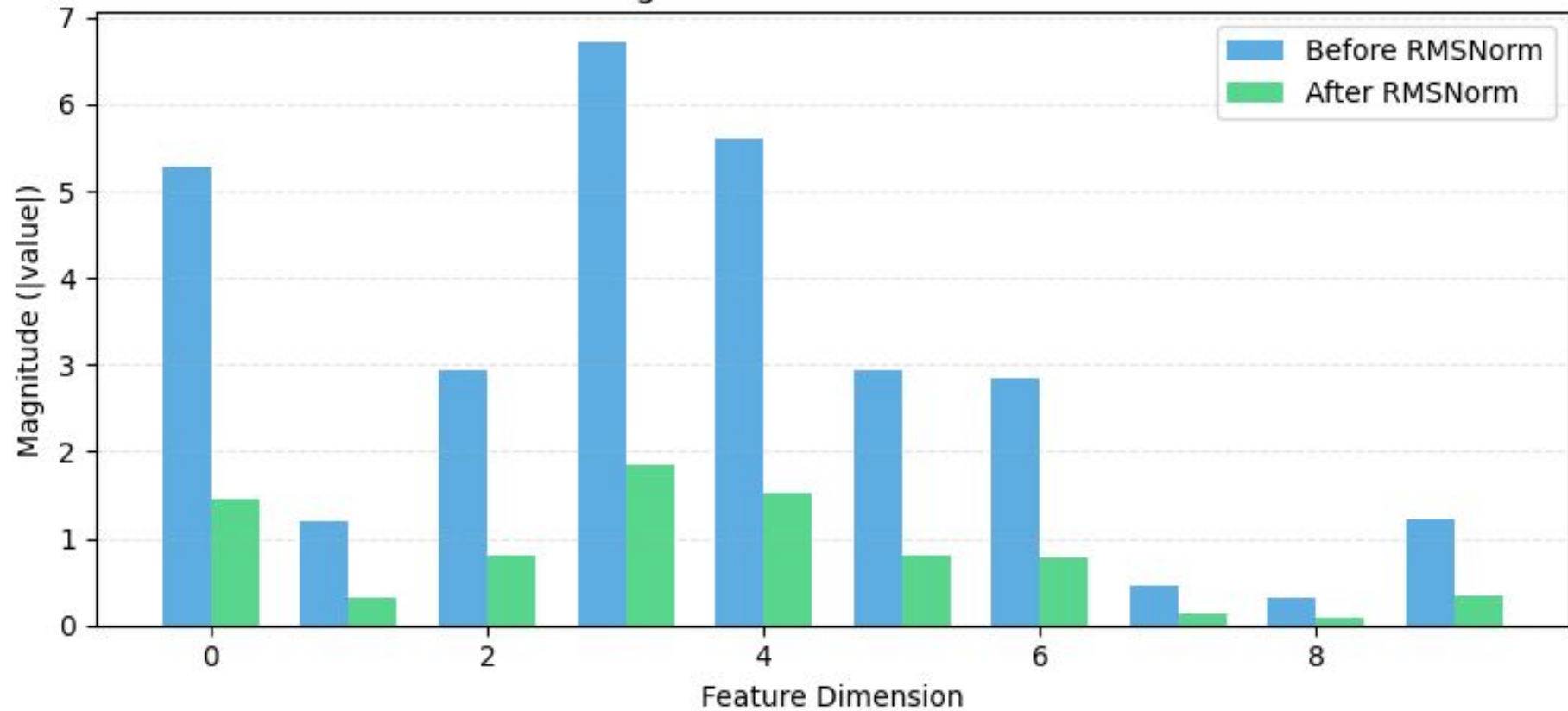
Normalize by magnitude only:

$$\text{RMSNorm}(x) = \gamma * x / (\text{rms}(x) + \epsilon)$$

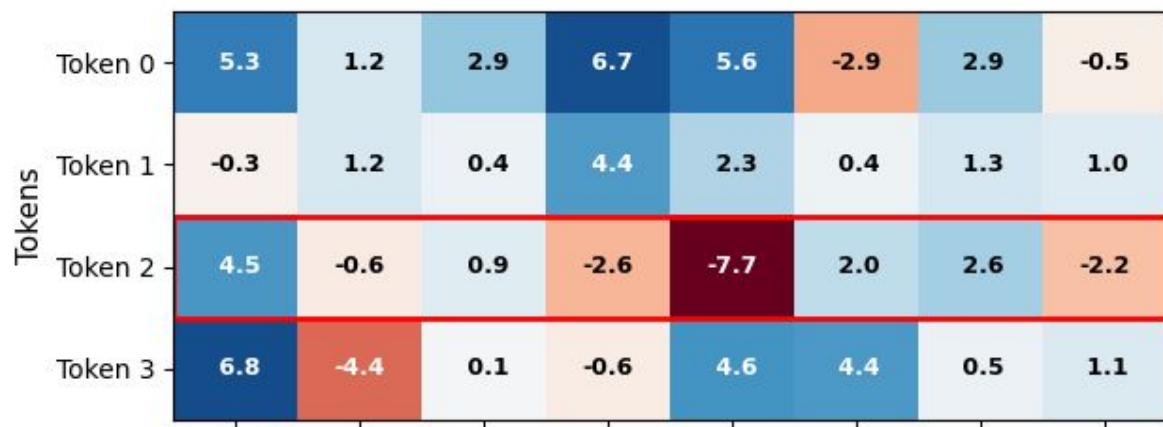
Key difference:

- LayerNorm centers by mean and scales by std → more accurate
- RMSNorm skips mean subtraction → faster, nearly identical stability

Feature Magnitudes Before vs After RMSNorm



Before RMSNorm



← Normalize this whole row (one token)

After RMSNorm



Pre-LN vs Post-LN



Post-LN (original Transformer): LayerNorm after each attention + residual

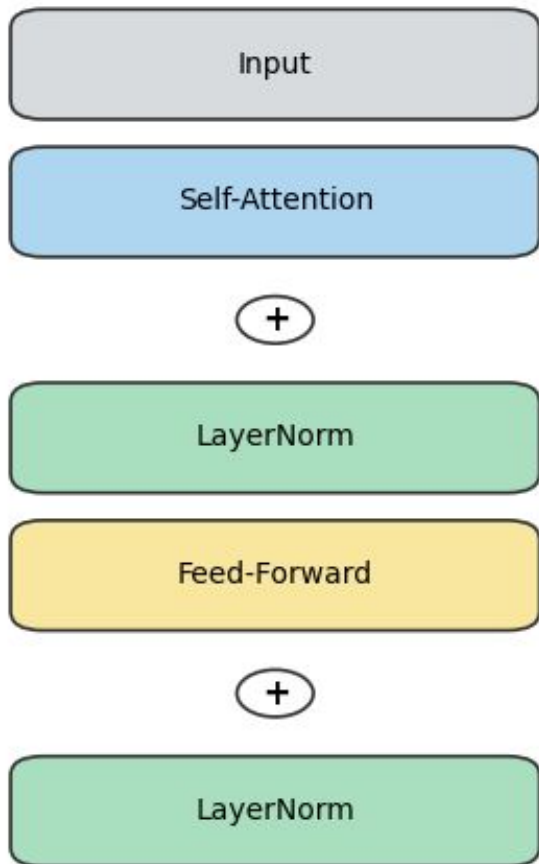
- Decent quality, but hard to train deep models

Pre-LN (modern LLMs): LayerNorm before attention + feed-forward

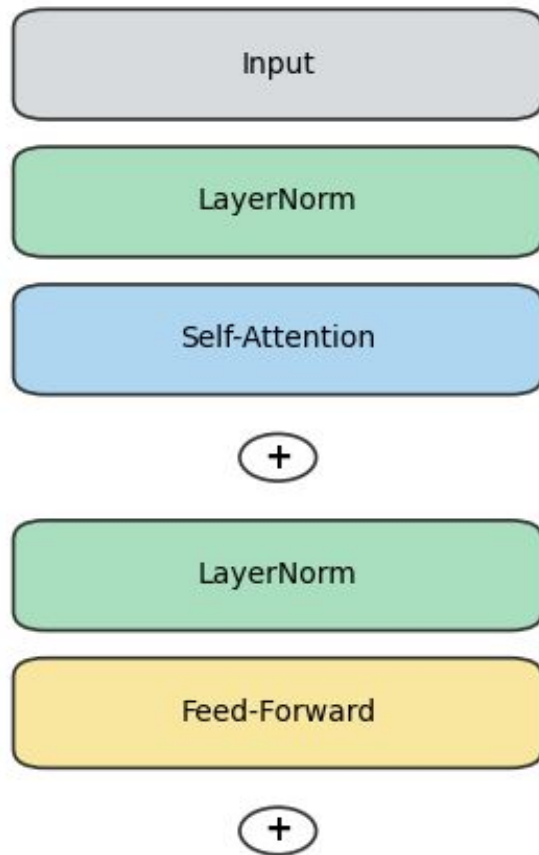
- Keeps activations within a stable range before large matrix multiplications
- Prevents softmax saturation by controlling the scale of Q and K vectors
- Makes training converge faster and more reliably

Adopted by most modern LLMs: GPT, LLaMA, PaLM, Mistral.....

Post-LN (Original Transformer)



Pre-LN (Modern LLMs)



More Tricks ⚡



AdamW (with weight decay on non-sparse weights)

→ Adam + decoupled weight decay → smoother convergence, better generalization

Learning rate warm-up + cosine decay

→ Gradually increases LR early to prevent divergence, then smoothly decays to stabilize convergence

Gradient clipping

→ Caps extreme gradient values to prevent sudden parameter spikes and exploding updates

Gradient accumulation

→ Accumulates gradients over multiple micro-batches to simulate a larger effective batch size

Takeaways



✓ Modern LLM De-facto Setup:

Pre-LN + RMSNorm + AdamW + LR Warm-up



Model Family

From Building Blocks to Model Families



The same Transformer block can act very differently depending on:

- **Masking pattern** – what each token can “see”
- **Training objective** – what the model is asked to predict

These choices define 3 major model families:

- **Decoder-only** → generates text step by step
- **Encoder-only** → understands full context
- **Encoder-Decoder** → translates one sequence into another

Decoder-only Family – Language Model (LM)



Mask: **Causal** → token t can observe only tokens $< t$

Objective: **Predict next token** → $P(\text{token}_{\square} \mid \text{tokens}_{(1 \dots \square-1)})$

Usage:

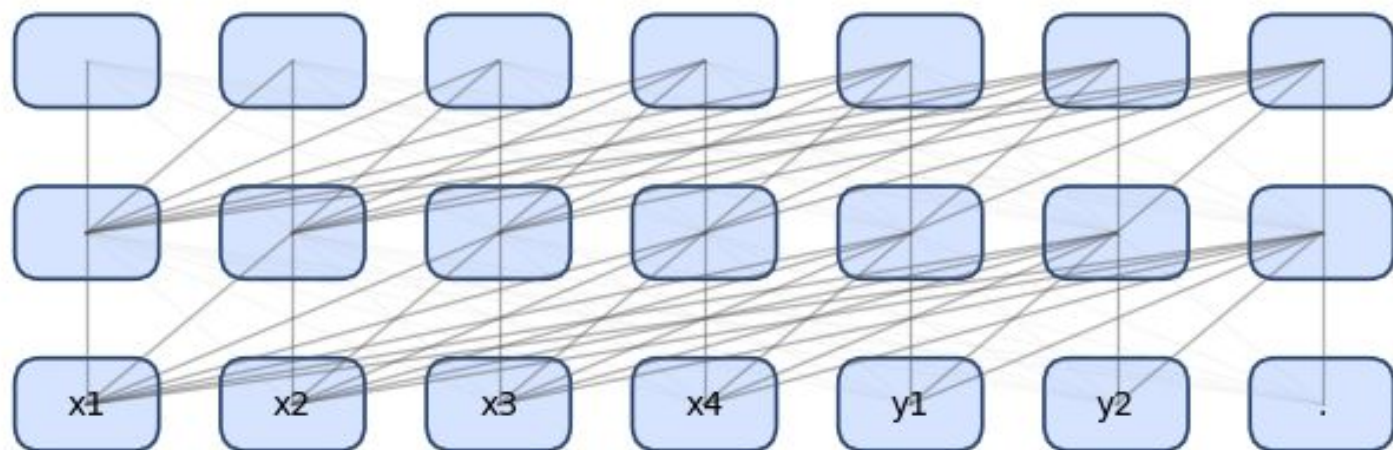
- Natural language generation (chatbots, storytelling)
- Code generation and completion (Copilot, Claude)

Examples:

GPT-2 / GPT-3 / GPT-4, LLaMA, Gemini, Mistral....

Decoder-only

causal self-attention



Encoder-only Family – Masked Language Model (MLM)



Mask: **Full attention** → all tokens can attend bidirectionally

Objective: Predict **randomly masked tokens** using both-sided context

Usage:

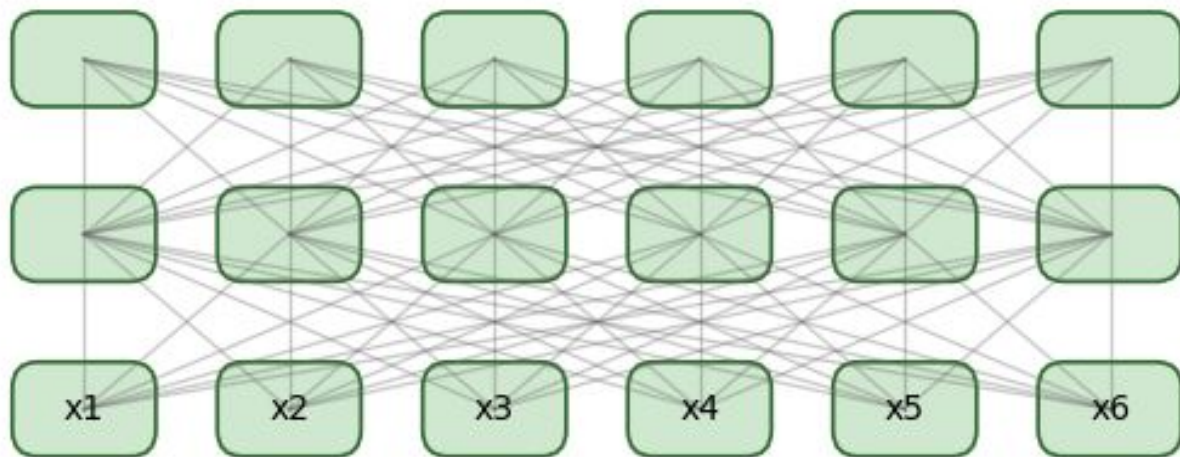
- Text classification (sentiment, topic)
- Retrieval and ranking (semantic search, dense embeddings)

Examples:

BERT, RoBERTa, SpanBERT, ALBERT.....

Encoder-only

bi-directional self-attention



Encoder-Decoder Family – Seq2Seq Denoising



Mask:

- Encoder → Full attention (understanding input)
- Decoder → Causal mask (generate token by token)
- **Cross-attention** → connects decoder queries to encoder outputs

Usage:

- Machine translation
- Image → text generation

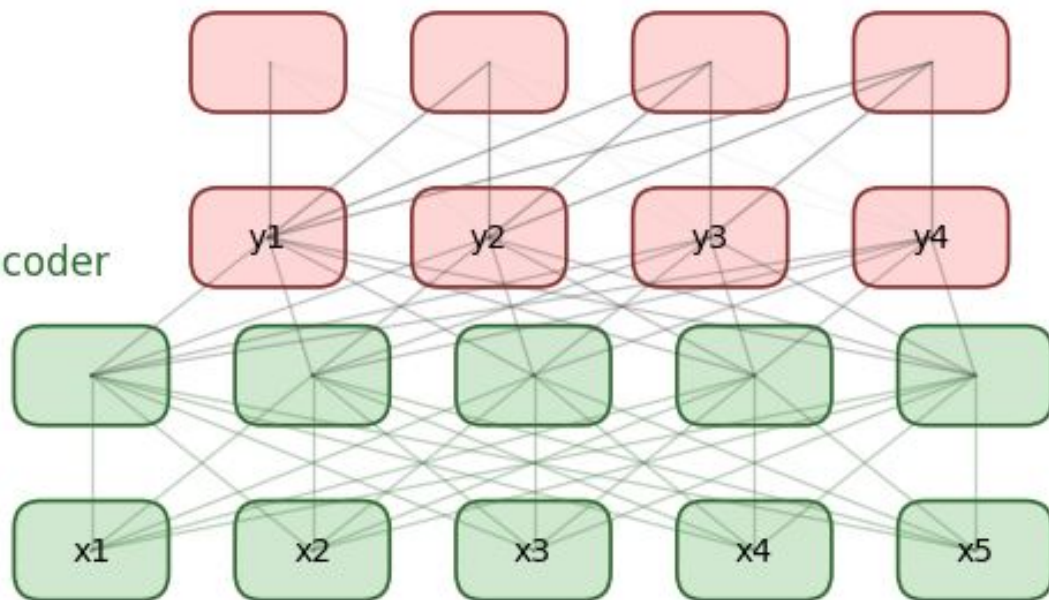
Examples:

- T5, BART, mT5, MarianMT.....

Encoder-Decoder (seq2seq)

Decoder

Encoder



Takeaways



Free text corpus → Decoder-only (Language Modeling)

- No paired data, just raw text
- Train to predict the next token - learns grammar, style, and coherence

Feature learning or retrieval → Encoder-only (Masked LM)

- Focus on deep understanding, not generation
- Learns contextual embeddings for classification or search

Labeled input/output pairs → Encoder–Decoder (Seq2Seq)

- Each input has a corresponding target (translation, summarization)
- Model learns conditional mapping $P(\text{output} \mid \text{input})$



Decoding Strategies

Decoding Controls the Model's Voice



The model outputs a **probability distribution** over the vocabulary at every step

Decoding = the decision rule for turning probabilities into actual words

This choice directly affects:

- **Quality** → coherence, grammatical correctness
- **Diversity** → creativity, surprise, and variability

Different decoding strategies can completely change tone and creativity

Decoding Controls the Model's Voice



Greedy decoding

- Always picks the most probable next token

Beam search

- Keeps several candidate sequences and expands the best-scoring ones

Sampling (Top-k / Top-p)

- Randomly sample from the most likely tokens

Greedy Decoding



Always pick the token with **the highest probability** at each step

At step t , choose:

$$y_t = \operatorname{argmax} P(y \mid y_1, y_2, \dots, y_{t-1})$$

The next token y_t is appended to the sequence, then repeats

Fast and deterministic \rightarrow same prompt **always yields the same output**

Can fall into repetitive loops if top token probability dominates

Best for short factual answers or structured text generation

Beam Search 🌲



Keeps multiple candidate sequences (beams) at each step

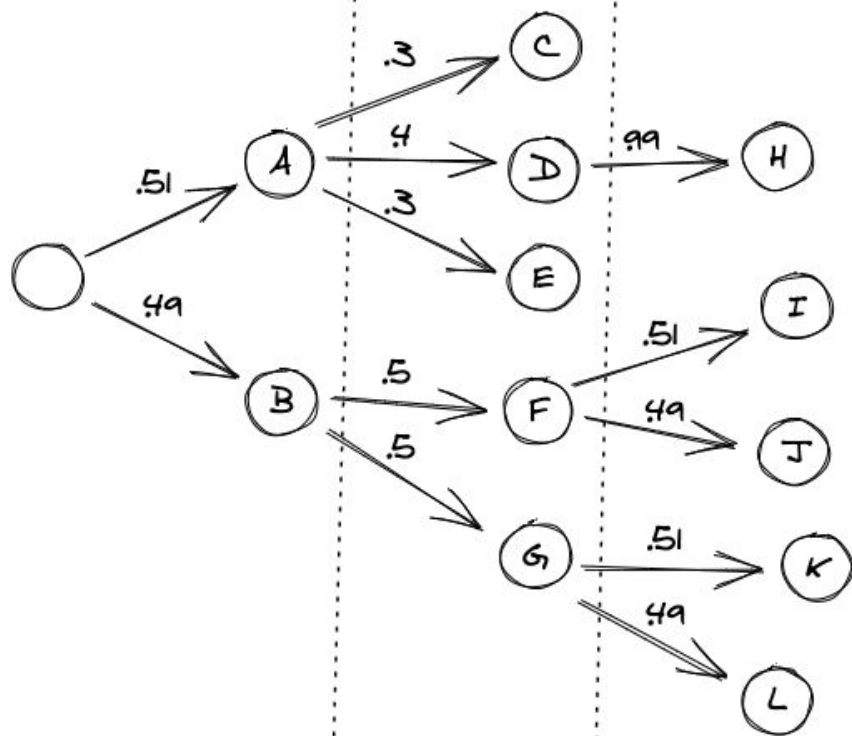
At step t , select top_beam sequences with the highest cumulative log-probability:

$$\text{Score}(y_1, \dots, y_t) = \sum \log P(y_t \mid y_1, \dots, y_{t-1}, x)$$

Produces more coherent and complete results than greedy decoding

Lower randomness \rightarrow less diversity, slower inference

Works best for translation, summarization, and other structured tasks



Greedy: A

Beam: A, B

Greedy: AD

Beam: BF, BG

Greedy: ADH

Beam: BFI, BGK

Sampling Methods (Top-k and Top-p)

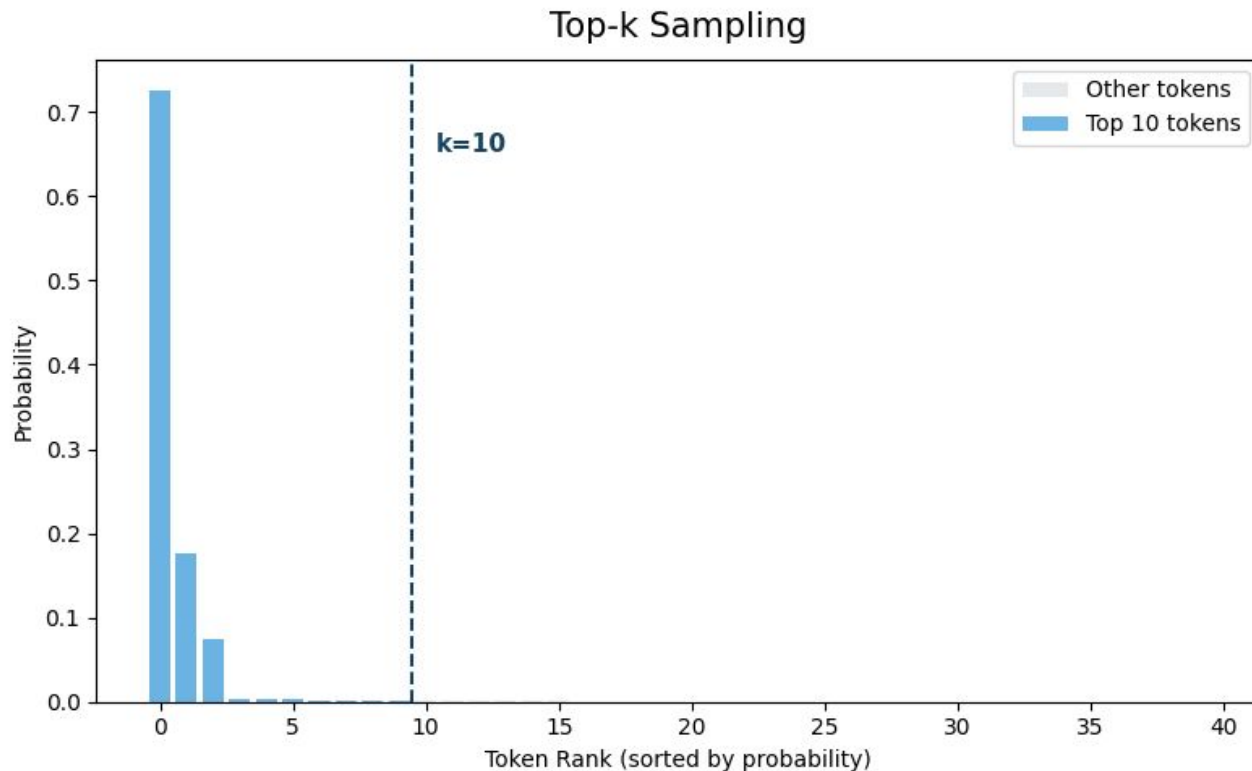


Top-k: sample from the **k most probable** tokens (e.g., $k = 50$)

Top-p (Nucleus): dynamically sample from the smallest set whose **probabilities sum $\geq p$** (e.g., $p = 0.9$)

Introduces controlled randomness \rightarrow more natural and varied outputs

Sampling Methods (Top-k)



Takeaways



Greedy decoding → safe, focused, and deterministic

- Always picks the top token → consistent but can repeat or lack creativity

Beam search → structured and globally optimized

- Evaluates multiple candidate paths → improves coherence, often used in translation or summarization

Sampling (Top-k / Top-p) → diverse and expressive

- Introduces controlled randomness → more natural, creative responses at the cost of stability



Research Frontier



Long-Context

Long-Context Transformers



Problem

Standard attention cost is $O(n^2)$ → too slow for long documents or sessions

Motivation

- Modern LLMs need to reason across 10k+ tokens: long conversations, multi-document reasoning, full-context summarization
- Long context enables more natural interactions and persistent memory

Common solutions

Sliding-Window attention, Sparse attention

Sliding Window Attention



Divides long sequences into **local chunks** (windows)

Each token attends only to nearby tokens **within a fixed window size**

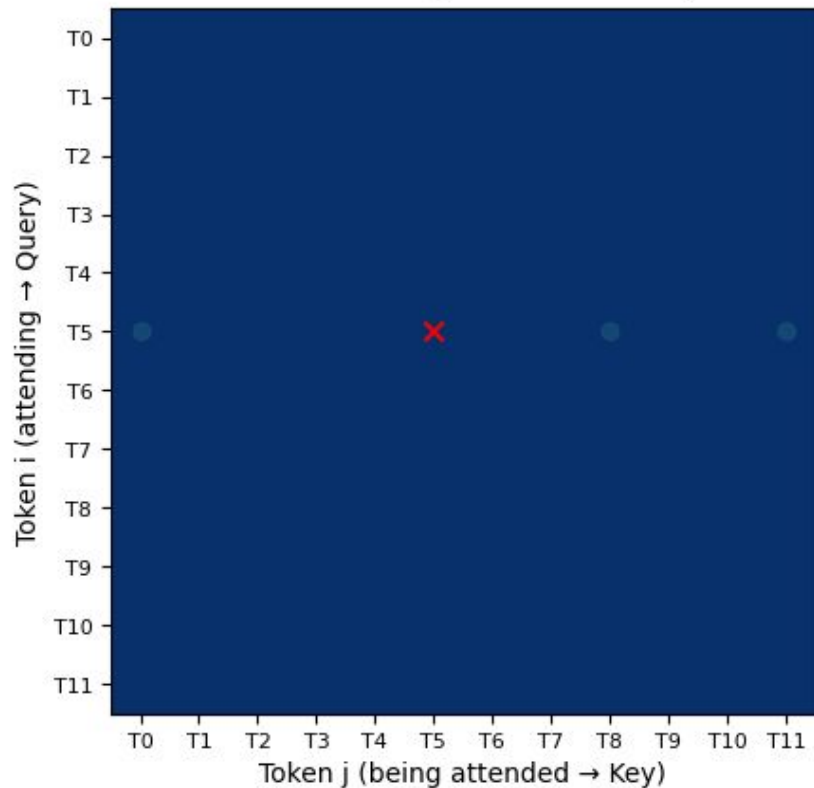
Windows may overlap slightly to preserve continuity across boundaries

Great for long documents - balances efficiency and context

Used in models like Longformer, Transformer-XL, LLaMA.....

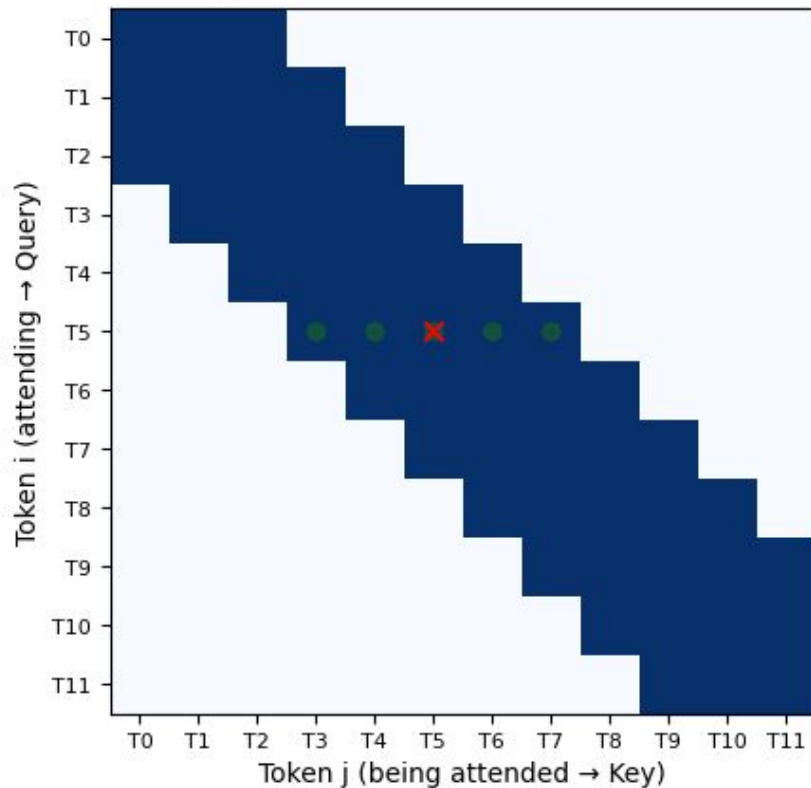
Each token attends to ALL others $\rightarrow O(n^2)$

Full Attention (Every token \leftrightarrow Every token)



Each token attends only nearby tokens $\rightarrow O(n \cdot w)$

Sliding / Local Attention (Window = ± 2)



FlashAttention ⚡



Core Idea:

- Standard attention connects every token with every other token → $O(n^2)$ cost
- Sparse attention keeps only the important links — making it much faster

How It Works:

- Local pattern: each token attends only to its neighbors
- Global pattern: a few special tokens (e.g., [CLS], separators) can see everything → summarize or broadcast information
- Random / Strided pattern: connect to a few distant tokens → let the model “jump” occasionally for long-range info

Evaluating Long Context Models



Benchmarks

Needle-in-a-Haystack: insert a single fact in thousands of irrelevant tokens

What to Measure

Retrieval Accuracy: can the model locate the right span or entity inside 10k+ tokens?

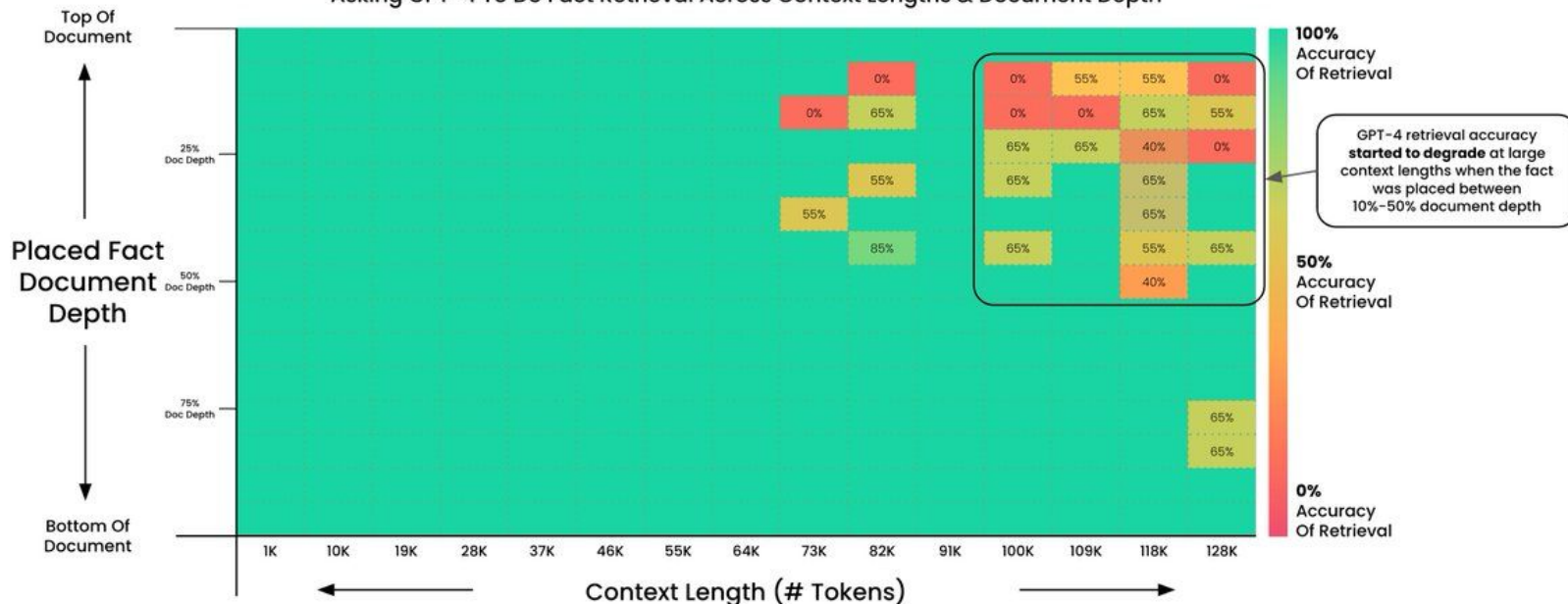
Faithful Generation: does it use retrieved information correctly without hallucinating?

Evaluating Long Context Models



Pressure Testing GPT-4 128K via "Needle In A HayStack"

Asking GPT-4 To Do Fact Retrieval Across Context Lengths & Document Depth



Goal: Test GPT-4 Ability To Retrieve Information From Large Context Windows

A fact was placed within a document. GPT-4 (1106-preview) was then asked to retrieve it. The output was evaluated for accuracy.

This test was run at 15 different document depths (top > bottom) and 15 different context lengths (1K > 128K tokens).

2x tests were run for larger contexts for a larger sample size.



Mixture-of-Experts (MoE)

Mixture-of-Experts (MoE)



Each token activates **only a small subset (top-k) of experts**

Greatly increases capacity without raising compute cost

Router network decides which experts process each token

Experts are trained jointly and specialize on different patterns

Used in Switch Transformer, Mixtral, Llama and recent scalable LLMs

Routing and Balancing



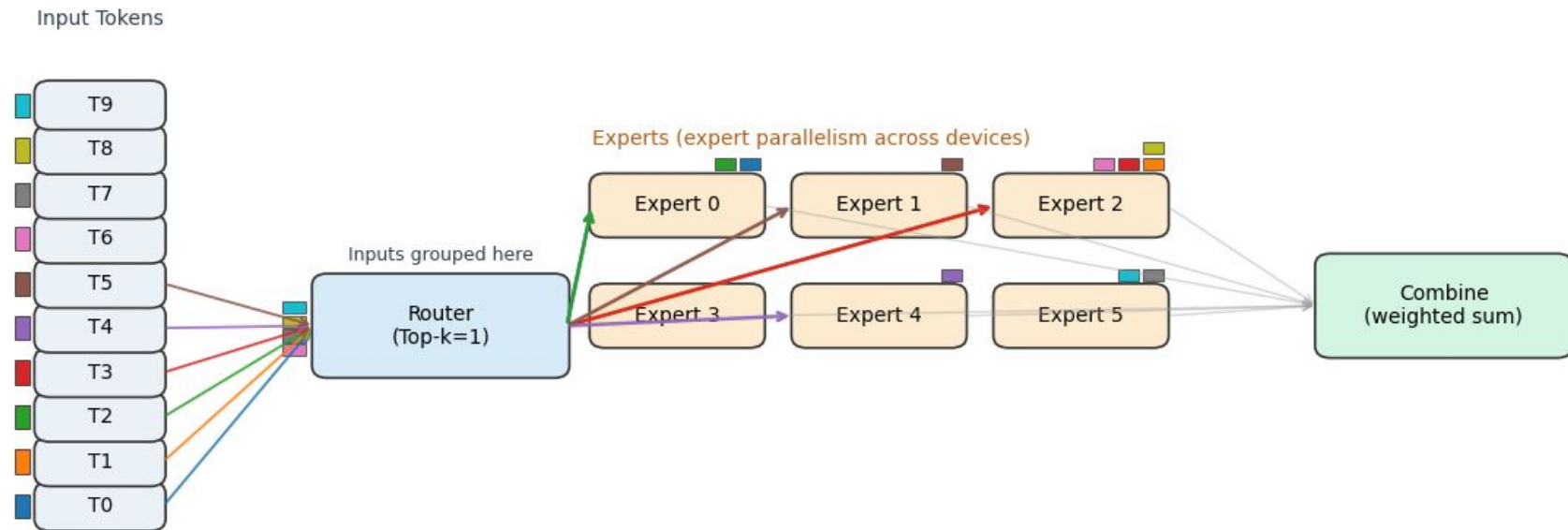
Router predicts which experts to use for each token

- Predicts which experts each token should use (usually top-1 or top-2)
- Implemented as a lightweight gating network - outputs routing probabilities

Load balancing loss prevents all tokens from choosing same experts

- Ensures tokens are distributed evenly across experts
- Prevents “expert collapse,” where all tokens pick the same expert

Transformer MoE (Training) — Token Routing to Experts (Top-k)



Research Trend – Modular Reasoning Systems



MoE encourages specialization

- Each expert implicitly learns to handle specific domains (math, reasoning)

Goal: scalable systems that can grow capacity by adding new experts without retraining the entire model

Example directions:

- Task-aware expert activation (activate “math” or “code” experts)
- Hierarchical MoE – group experts into thematic clusters
- Continual learning with plug-in experts