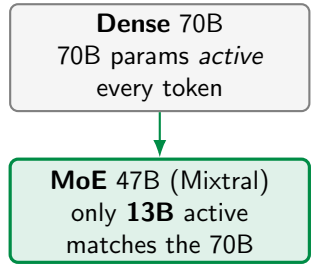
# What if a model had 10x the parameters at the same cost?

A bigger model is a better model – but every extra parameter costs FLOPs on *every* token. That is the wall dense scaling hits.

**Mixture of Experts (MoE)** breaks the link. Give a layer $N$ specialist sub-networks ("experts") and a tiny **router** that sends each token to only $k$ of them.

Parameters grow with $N$; compute-per-token is fixed by $k$. So you get a **much bigger** model for *roughly the same* cost per token.

Today: the core idea, the **Switch Transformer** (2021, scaled to 1.6T params), and **Mixtral 8x7B** (2024) – a 47B model that runs like a 13B one.




same quality, a fraction of the compute

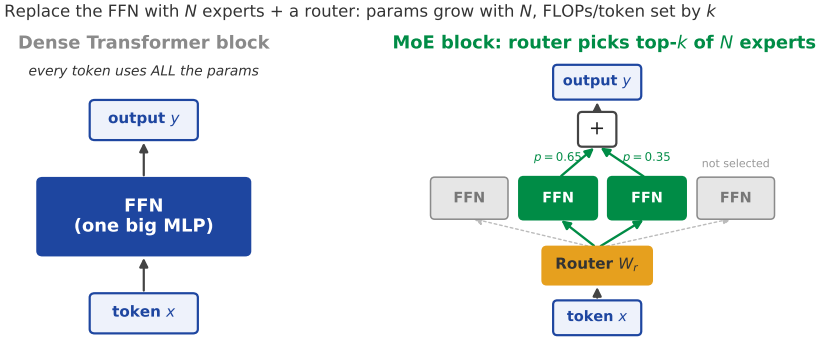
# Outline

## Decouple capacity from compute

Most of a Transformer's parameters live in the FFN. Split it into experts.

# Replace the FFN with $N$ experts and a router

A Transformer block's FFN holds **most of its parameters**. MoE swaps that one FFN for $N$ **parallel experts** plus a small **router** that sends each token to only a few.




A dense block runs *every* token through *all* the params; an MoE block runs each token through only its **top**-$k$ experts – so total params can be huge while compute-per-token stays small.

# The routing math

A router $W_r$ turns token $x$ into a distribution over the $N$ experts; keep the **top-$k$**:

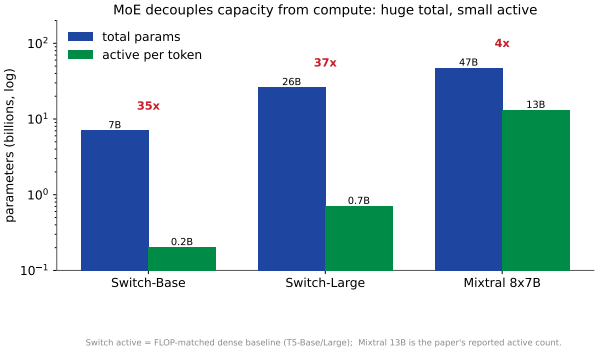
$$p_i(x) = \frac{e^{(W_r x)_i}}{\sum_{j=1}^{N} e^{(W_r x)_j}}, \qquad \boxed{y = \sum_{i \in \mathcal{T}} p_i(x)\, E_i(x)}$$

with $\mathcal{T}$ = top-$k$ indices and $E_i$ expert $i$'s FFN.

- **Total params** $\approx N \times |\text{expert}| + \text{shared}$ – grows with $N$.
- **Compute/token:** only $k$ experts fire, FLOPs $\approx k \times |\text{expert}|$ – *independent of* $N$.
- The gate $p_i(x)$ keeps the router **differentiable**, so it is learned.

Switch Transformer uses $k = 1$; Mixtral uses $k = 2$. Same equation, different $k$ and $N$.

# Two parameter counts: total vs active




MoE forces a distinction the dense world never needed:

- **Total (sparse) params** – everything stored; grows with $N$. Sets the *memory* cost.
- **Active params** – what actually fires per token; set by $k$. Sets the *compute* cost.

The whole trick in one line: **total** $\gg$ **active**. Buy quality with params, pay compute only for $k$ of them.

# Switch Transformer

Make sparse routing *simple* and *stable* – then scale it to a trillion.

# Top-1 routing: the radical simplification

Earlier MoE work (Shazeer et al., 2017) assumed you *must* route to $k \geq 2$ experts – otherwise the router has no way to compare and get a gradient. Switch says: route to **exactly one** expert ($k = 1$).

**Why top-1 is a good deal:**

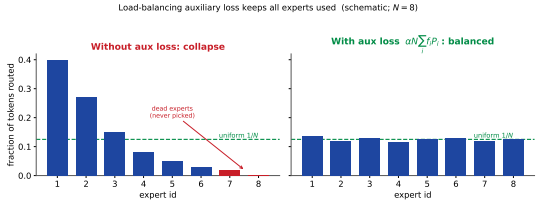
- **Router compute drops** – pick a single argmax, not a top-$k$ set.
- **Expert capacity halves** – each token lands in one place, so buffers shrink.
- **Communication is simpler** – less data shuffled between devices.

The gate value $p_i(x)$ still multiplies the chosen expert's output, so the router stays differentiable even at $k = 1$. Quality holds; everything else gets cheaper.

# The load-balancing auxiliary loss

Left alone, the router collapses onto a **few favourite experts**; the rest go **dead**. Switch adds a differentiable auxiliary loss per layer:

$$\text{loss} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \, P_i$$



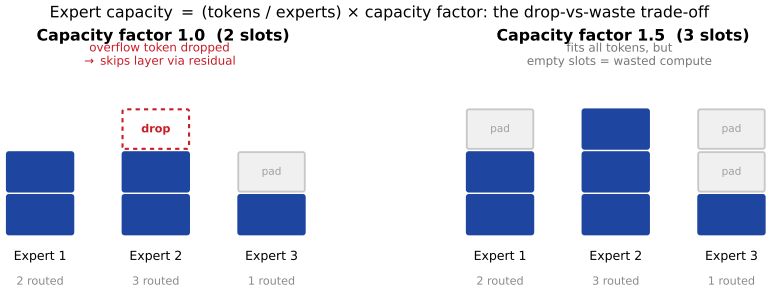

- $f_i$ = fraction of tokens *dispatched* to expert $i$; $P_i$ = its mean router *probability*.
- Minimized when both are $1/N$: **uniform** use.
- $\alpha = 10^{-2}$ – enough to balance, not enough to drown the main loss.

Balancing is not optional polish – without it, capacity is wasted on dead experts and the model trains far worse.

# Expert capacity and token dropping

Hardware wants **fixed** tensor shapes, so each expert gets a fixed number of slots:

$$\text{expert capacity} = \left( \frac{\text{tokens per batch}}{\text{number of experts}} \right) \times \text{capacity factor}$$




Capacity factor $> 1$ buys slack against uneven routing. Too low $\rightarrow$ **dropped tokens** (skip the layer via the residual). Too high $\rightarrow$ **padding** (wasted compute). Good balancing lets you keep it near 1.

# Expert parallelism and stable training

**Expert parallelism.** Put different experts on different devices. Adding devices adds experts – so **parameters scale with your hardware** while per-device memory and compute stay bounded.

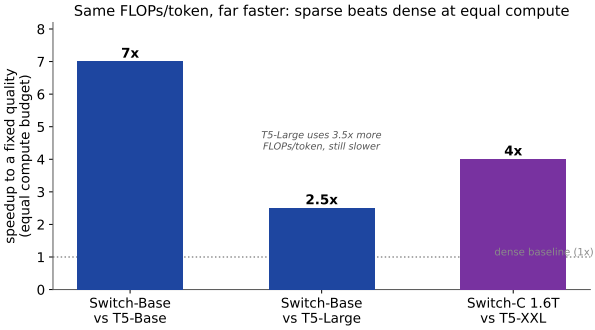
The price: an **all-to-all** shuffle each layer to send every token to its expert's device and bring the result back.

**Keeping it stable.** Sparse hard-routing is twitchy. Switch's fixes:

- **Selective precision** – cast just the router to float32. bfloat16 everywhere else diverged; this kept full speed.
- **Smaller init** – shrink the init scale 10×: better quality, much lower variance across seeds.
- **Expert dropout** – higher dropout *inside* experts when fine-tuning (they overfit small data).

Parameters now live *across* the cluster – more devices, more experts, same FLOPs per token.

# Does it pay off? Faster pretraining at equal compute




At a **fixed compute budget** (same FLOPs/token, same hardware, same wall-clock):

- Switch-Base reaches T5-Base quality **7x** faster.
- Still **2.5x** faster than T5-Large – which spends 3.5x more FLOPs/token.
- Scaled to **Switch-C, 1.6T params** (2048 experts): **4x** faster than T5-XXL.

More experts = more parameters = faster learning per FLOP. A new axis to scale on.

## The costs nobody puts on the headline

Sparsity is not free – it moves the cost from compute to memory and plumbing.

# Where MoE bites back

### Training / routing troubles

- **Load balancing** – needs the aux loss or experts die; still a tuning knob.
- **Routing instability** – hard argmax decisions make big models diverge (Switch-XXL was unstable even with the fixes).
- **All-to-all communication** – every layer shuffles tokens across devices; can dominate wall-clock at scale.

### The memory asymmetry

- Only $k$ experts *compute*, but **all $N$ must be stored** and loaded.
- Serving Mixtral costs memory for its **full 47B**, even though 13B run.
- So MoE saves *compute*, not *memory* – the opposite of what a newcomer expects.

"Cheaper" means cheaper **FLOPs per token**. You still pay full **memory**, extra **communication**, and real **engineering complexity**.

# Mixtral 8x7B

The idea, three years later, as a concrete open-weights model.

# Mixtral: a modern sparse MoE

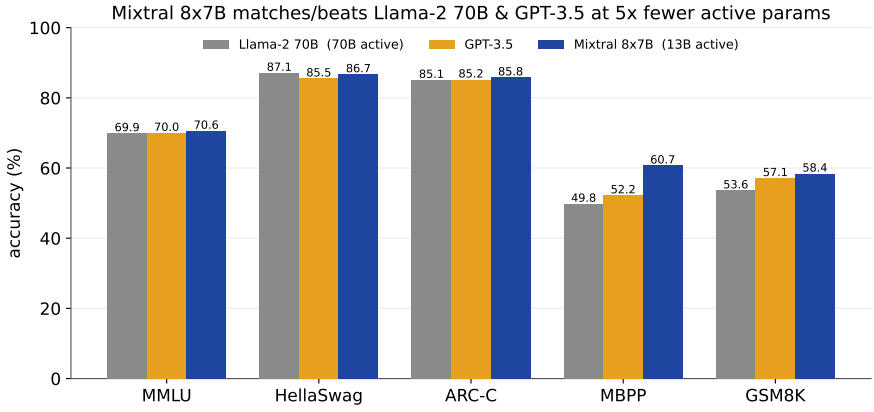
Same backbone as Mistral 7B, but each layer's FFN is replaced by a MoE layer:

- **8 experts** per layer, **top-2** routing ($N = 8$, $k = 2$).
- Experts are standard **SwiGLU** FFN blocks.
- **47B total** params, but only **13B active** per token.
- 32 layers, 32k-token context, released under Apache 2.0.

The router can pick a *different* pair of experts at every token and every layer – so 13B of compute draws on all 47B of stored knowledge.

| **Mixtral 8x7B** | **value** |
|---|---|
| dim | 4096 |
| layers | 32 |
| hidden dim (FFN) | 14336 |
| context length | 32768 |
| num. experts $N$ | 8 |
| top-$k$ | 2 |
| total params | 47B |
| **active params** | **13B** |

# Mixtral vs the dense heavyweights




With **13B active** params, Mixtral matches or beats **Llama-2 70B** (5x more active) and **GPT-3.5** across the board – and is *far* ahead on code (MBPP) and math (GSM8K). Mixtral 8x7B-Instruct then out-scored GPT-3.5 Turbo, Claude-2.1, and Gemini Pro on human evals.

# What do the experts actually specialize in?

Natural guess: expert 3 is "the math expert," expert 5 is "the code expert." The paper checked – and found **no clear topic specialization**.

- Across ArXiv, biology, and philosophy text, the expert-usage histograms look **almost identical** – routing is not by domain.
- Instead it tracks **syntax**: indentation tokens in code, or a repeated word like `self`, keep hitting the same expert.
- Strong **temporal locality** – consecutive tokens often share an expert (useful for caching, but risks over-subscribing a device).

"Experts" is a misnomer: they are not human-interpretable topic specialists. The router learns a low-level, syntactic partition – whatever helps the loss.

## So when does MoE win?

It is a compute-vs-memory trade, not a free lunch.

# MoE wins vs MoE costs

**Reach for MoE when...**

- You are **compute-bound** at scale and want more quality per FLOP/token.
- You serve **batched / high-throughput** traffic, where all-to-all overhead amortizes.
- You have the **memory** to store all experts (many devices).
- You want faster pretraining at a fixed compute budget.

**Think twice when...**

- You are **memory-bound** – you pay for all $N$ experts, not $k$.
- You need **single-request, low-latency** inference (routing + all-to-all overhead).
- You cannot afford the **engineering complexity** (balancing, sharding, capacity tuning).
- A distilled **dense** model would fit your box (Switch distills, keeping $\sim$30% of the gain).

Rule of thumb: MoE trades **memory + complexity** for **quality at a fixed compute-per-token**.

# Recap

- **MoE = experts + a router.** Replace a block's FFN with $N$ experts; a router sends each token to its **top-$k$**. Total params grow with $N$; FLOPs/token set by $k$: $y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x)$.
- **Switch Transformer**: top-1 routing, a load-balancing aux loss ($\alpha N \sum_i f_i P_i$), expert capacity / token dropping, expert parallelism – scaled to **1.6T** params and **7x** faster pretraining at equal compute.
- **Hard parts**: load balancing / dead experts, routing instability, all-to-all cost, and the memory asymmetry – **all** experts stored though only $k$ run.
- **Mixtral 8x7B**: 8 experts, top-2, **47B total / 13B active**; matches or beats Llama-2 70B and GPT-3.5 at 5x fewer active params. Experts specialize by **syntax**, not topic.
- **The trade**: MoE buys quality-per-FLOP with memory and complexity.

**Seen already:** *[LLM-8] Qwen3* is a modern MoE in production – the same router + top-$k$ experts you just met, at frontier scale.