ARCHITECTURE · CONDITIONAL COMPUTATION

# Mixture of Experts and top-k routing

A mixture of experts hands each input to a few sub-networks chosen by a router. The parameter count grows with the number of experts, but the computation per token depends only on $k$.

Advanced · 11 min · MoE · routing · top-k · sparsity · Example — not reviewed by the author

Sheet by Simon-Pierre Boucher

sheet.spboucher.ai/s/mixture-of-experts-top-k-routing
November 17, 2026

In an ordinary network, **all** parameters work on **every** input. A Mixture of Experts (MoE) breaks that rule: a small gating network picks, for each input, a few specialised sub-networks and leaves the others idle [3]. The idea goes back to the 1990s [1]; its sparse version is the one found in large language models.

### Experts

$N$ sub-networks of the same shape but with different parameters. In a Transformer, each one is a feed-forward network.

### A router

A very light network that scores every expert from the input, then keeps only $k$ of them.

### A balancing term

An auxiliary loss that pushes the router to spread the work; without it a few experts receive everything.

### Score

One matrix multiplication yields $N$ scores. Its cost is negligible next to that of one expert.

### Select

$k$ experts are kept, with $k \ll N$. Some models route each token to a single expert [5].

### Compute

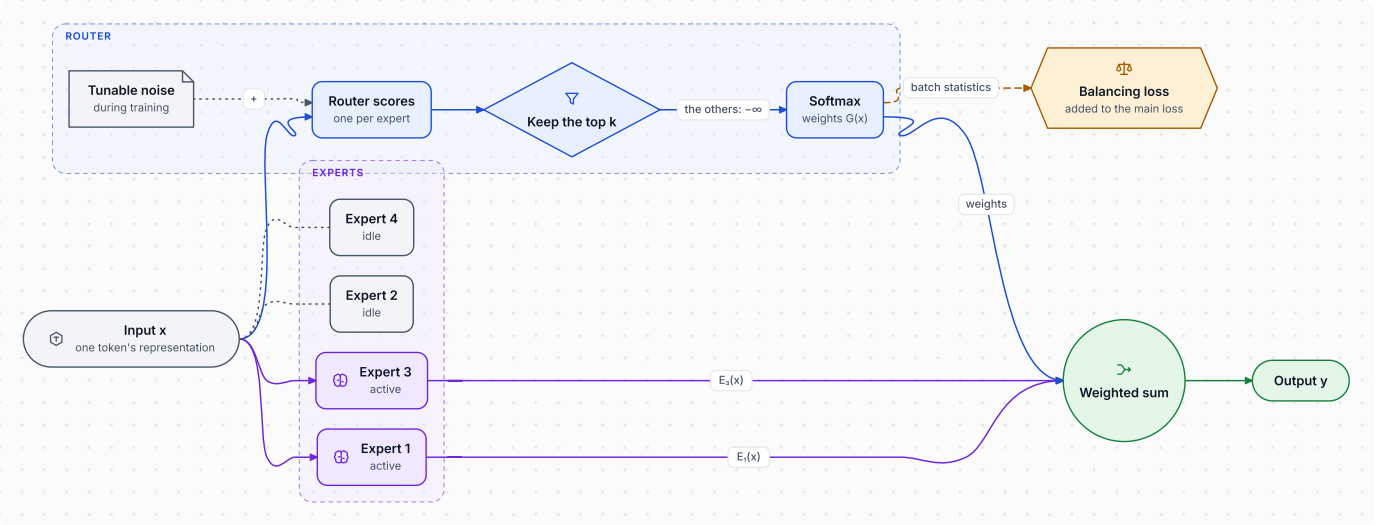
Only the selected experts run, each on the tokens it was assigned.

### Recombine

The outputs are weighted and summed, then the next layer carries on as in an ordinary Transformer.

## 01 Anatomy of an MoE layer

**Top-k routing with N = 4 experts and k = 2**




Routing decision · Computation done by an active expert · Idle expert: path not taken, no computation · Auxiliary training signal · Output of the layer

1 **Score every expert**
— The router multiplies the input by a learned matrix $W_g$: one score per expert. During training, noise whose magnitude is itself learned is added.

2 **Keep only k experts**
— The top $k$ scores are kept, the others are set to $-\infty$. After the softmax their weight is exactly 0.

3 **Evaluate only the selected experts**
— The input is sent to experts 1 and 3 only. Experts 2 and 4 compute nothing: this is where the saving comes from.

4 **Combine the outputs**
— The output is the sum of the active experts' outputs, weighted by the router's weights.

5 **Watch the distribution**
— Over the whole batch, the load received by each expert is measured; an auxiliary loss penalises imbalance.

Only the experts kept by the router are evaluated; their outputs are combined with the router's weights [3].

Sources: 3

## 02 Routing in equations

### ƒ OUTPUT OF THE LAYER

$$y = \sum_{i=1}^{N} G(x)_i \, E_i(x)$$

Whenever $G(x)_i = 0$ there is no need to compute $E_i(x)$: the sparsity of the vector $G(x)$ translates directly into computation avoided [3].

| | |
|---|---|
| $x$ | Input of the layer (one token's representation) |
| $E_i$ | Expert $i$, a neural network with its own parameters |
| $G(x)_i$ | Weight the router gives to expert $i$; the weights sum to 1 |
| $N$ | Number of experts |

Sources: 3

### ƒ NOISY TOP-K GATING

$$G(x) = \mathrm{Softmax}\big(\mathrm{KeepTopK}(H(x), k)\big)$$

$$H(x)_i = (x \cdot W_g)_i + \varepsilon_i \cdot \mathrm{Softplus}\big((x \cdot W_{\text{noise}})_i\big), \quad \varepsilon_i \sim \mathcal{N}(0,1)$$

$$\mathrm{KeepTopK}(v,k)_i = \begin{cases} v_i & \text{if } v_i \text{ is among the top } k \text{ elements of } v \\ -\infty & \text{otherwise} \end{cases}$$

The noise, whose per-expert magnitude is learned, helps spread the load during training [3].

| | |
|---|---|
| $W_g$ | Weight matrix of the router |
| $W_{\text{noise}}$ | Learned matrix setting the noise magnitude per expert |
| $k$ | Number of experts activated per input |
| $\mathrm{Softplus}$ | Function $z \mapsto \ln(1 + e^z)$, always positive |

Sources: 3

DEFINITION

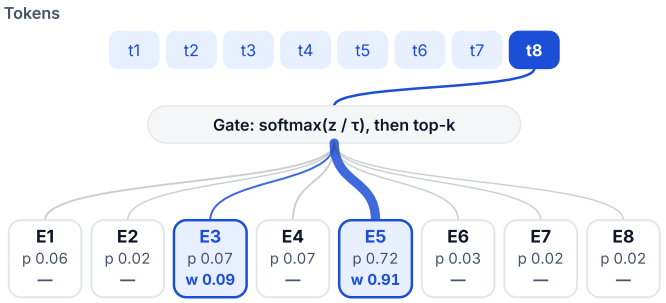
### Where does the MoE layer sit?

In expert-based Transformers the MoE layer **replaces the feed-forward network** of some blocks; attention stays dense. Routing is done token by token, so two tokens of the same sentence may go through different experts [4] [5].

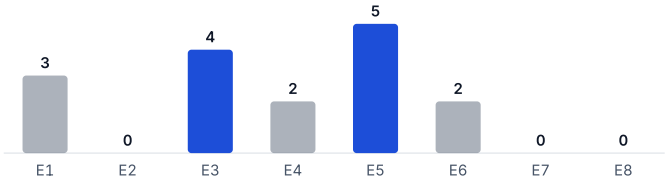
### Route tokens to experts

Synthetic data — seed 11

For each token the gate computes a softmax over $N$ experts, keeps only the $k$ most probable and renormalises their weights; the output is the weighted sum of those $k$ experts. Simplification: gate logits are random draws (a fixed preference per expert plus a token-specific term) rather than the output of a trained router.




Cumulative load: tokens routed to each expert




| Current token | Selected experts | Gate mass kept | Gate entropy (normalised) |
|---|---|---|---|
| t8 (8 / 8) | E5, E3 | 0.788 | 0.525 |

Final imbalance (max / mean): 2.5

Parameters: Experts N = 8; Active experts k = 2; Tokens = 8; Temperature τ = 1

Teaching simulation on randomly drawn scores (fixed seed): this is not the router of a trained model.

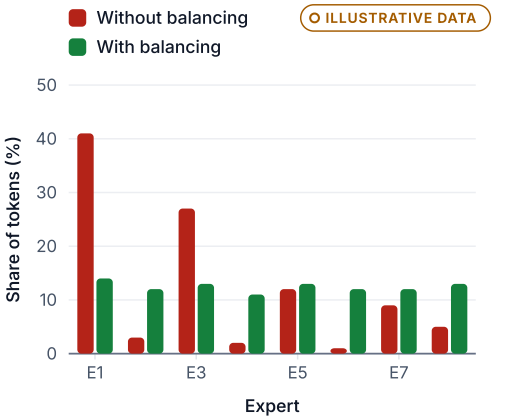
## 03 Balancing the load

WARNING

### The trap: routing collapse

Left unconstrained, the router tends to send almost everything to the same experts. The imbalance is self-reinforcing: favoured experts are trained more, so they get selected even more often [3]. The others become dead weight, and the overloaded experts turn into a hardware bottleneck.

### Share of tokens received by each expert




Illustrative values chosen by hand to show the shape of the phenomenon; these are not measurements.

### ƒ IMPORTANCE LOSS (SHAZEER ET AL.)

$$\mathrm{Importance}(X) = \sum_{x \in X} G(x)$$

$$L_{\text{importance}}(X) = w_{\text{importance}} \cdot \mathrm{CV}\big(\mathrm{Importance}(X)\big)^2$$

The loss is zero when all experts receive the same total importance over the batch, and grows with the dispersion [3].

| | |
|---|---|
| $X$ | Batch of training examples |
| $\mathrm{CV}$ | Coefficient of variation: standard deviation divided by the mean |
| $w_{\text{importance}}$ | Weight of the auxiliary loss, tuned by hand |

Sources: 3

### ƒ AUXILIARY LOSS (SWITCH TRANSFORMER)

$$L_{\text{aux}} = \alpha \cdot N \cdot \sum_{i=1}^{N} f_i \cdot P_i$$

Under a uniform distribution, $f_i = P_i = 1/N$ and the sum reaches its minimum $1/N$. Since $f_i$ is not differentiable, the gradient flows through $P_i$ [5].

| | |
|---|---|
| $f_i$ | Fraction of the batch's tokens dispatched to expert $i$ |
| $P_i$ | Mean probability the router assigns to expert $i$ over the batch |
| $\alpha$ | Coefficient of the auxiliary loss |

Sources: 5

## 04 Why it scales

**PARAMETERS AND COMPUTE DECOUPLE**

$$\underbrace{P_{\text{MoE}} \approx N \cdot P_E}_{\text{parameters}} \qquad \underbrace{C_{\text{token}} \approx k \cdot C_E + C_{\text{router}}}_{\text{compute per token}}$$

$f_x$ Result derived in this sheet

Relation derived from the definition of the layer: adding experts raises $N$, hence capacity, without touching $k$, hence the computation per token. It ignores the cost of communication between machines.

| | |
|---|---|
| $P_E$ | Number of parameters of one expert |
| $C_E$ | Compute cost of one expert for one token |
| $C_{\text{router}}$ | Cost of the router, small compared with $C_E$ |

**Three ways of using $N$ sub-networks.**

| | Dense layer | Soft mixture (1991) | Top-k MoE (SPARSE) |
|---|---|---|---|
| EXPERTS EVALUATED PER INPUT | a single network, always | all $N$ | only $k$ |
| COMPUTE PER TOKEN | $C_E$ | $N \cdot C_E$ | $k \cdot C_E$ |
| PARAMETERS | $P_E$ | $N \cdot P_E$ | $N \cdot P_E$ |
| SPECIFIC DIFFICULTY | none | cost grows with $N$ | balancing, non-differentiable selection |

**VERDICT**

Top-k MoE keeps the capacity of the soft mixture at a compute cost close to that of a dense layer. This decoupling is what made it possible to train models with several hundred billion parameters [4].

**ASSUMPTION**

### What sparse computation does not save

All experts must stay **loaded in memory**, since any of them may be called for the next token. An MoE model saves computation per token, not memory: for the same parameter count it takes as much room as a dense model.

## 05 Check your understanding

**QUESTION**

**An MoE layer goes from 8 to 64 experts while keeping $k = 2$. What happens, roughly?**

A. The layer's parameters are multiplied by 8; computation per token barely changes.

B. Both parameters and computation per token are multiplied by 8.

C. Nothing changes, since only 2 experts are used.

SHOW THE ANSWER

**A — Correct.** Correct: parameters follow $N$, compute follows $k$. Only the very light router computes 64 scores instead of 8.

**B — Incorrect.** That would be true of a soft mixture, where every expert is evaluated.

**C — Incorrect.** Two experts per token, but not always the same two: all 64 exist and must be stored and trained.

**Are experts specialised by subject (medicine, law, code)?**

Not by design. Nobody assigns roles to experts: the division of labour emerges from training, and routing is decided token by token from the internal representation. The word “expert” names a sub-network, not a human-readable skill.

**How is this different from an ensemble of models?**

An ensemble evaluates **all** models and averages their answers. An MoE is a single model, trained end to end, in which the router picks **a few** sub-networks inside one layer; the rest of the network is shared.

**How does the router learn if top-k selection is not differentiable?**

The gradient flows through the weights of the **selected** experts: those weights are a differentiable function of the scores. Discarded experts get no gradient for that example, hence the usefulness of noise and of the balancing loss so that each gets tried [3].

**Where does the idea come from?**

In 1991 Jacobs, Jordan, Nowlan and Hinton proposed a system of experts made to compete by a gating network, which learns to split a task into sub-tasks [1]. Jordan and Jacobs extended it in 1994 to a hierarchy of experts [2]. The large-scale sparse version dates from 2017 [3].

## References


[1] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, Geoffrey E. Hinton (1991). *Adaptive Mixtures of Local Experts.* Neural Computation, 3(1), 79–87. doi:10.1162/neco.1991.3.1.79

[2] Michael I. Jordan, Robert A. Jacobs (1994). *Hierarchical Mixtures of Experts and the EM Algorithm.* Neural Computation, 6(2), 181–214. doi:10.1162/neco.1994.6.2.181

[3] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, Jeff Dean (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.* arXiv:1701.06538. arxiv.org/abs/1701.06538

[4] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, Zhifeng Chen (2020). *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding.* arXiv:2006.16668. arxiv.org/abs/2006.16668

[5] William Fedus, Barret Zoph, Noam Shazeer (2021). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.* arXiv:2101.03961. arxiv.org/abs/2101.03961