# Mixture of Experts (MoEs) (8x7B)

# What is Mixture of Experts?

Mixture of Experts (MoE) is a technique that uses many different sub-models (or "experts") to improve the quality of LLMs.

Two main components define a MoE:

- **Experts** - Each FFNN layer now has a set of "experts" of which a subset can be chosen. These "experts" are typically FFNNs themselves.
- **Router** or **gate network** - Determines which tokens are sent to which experts.

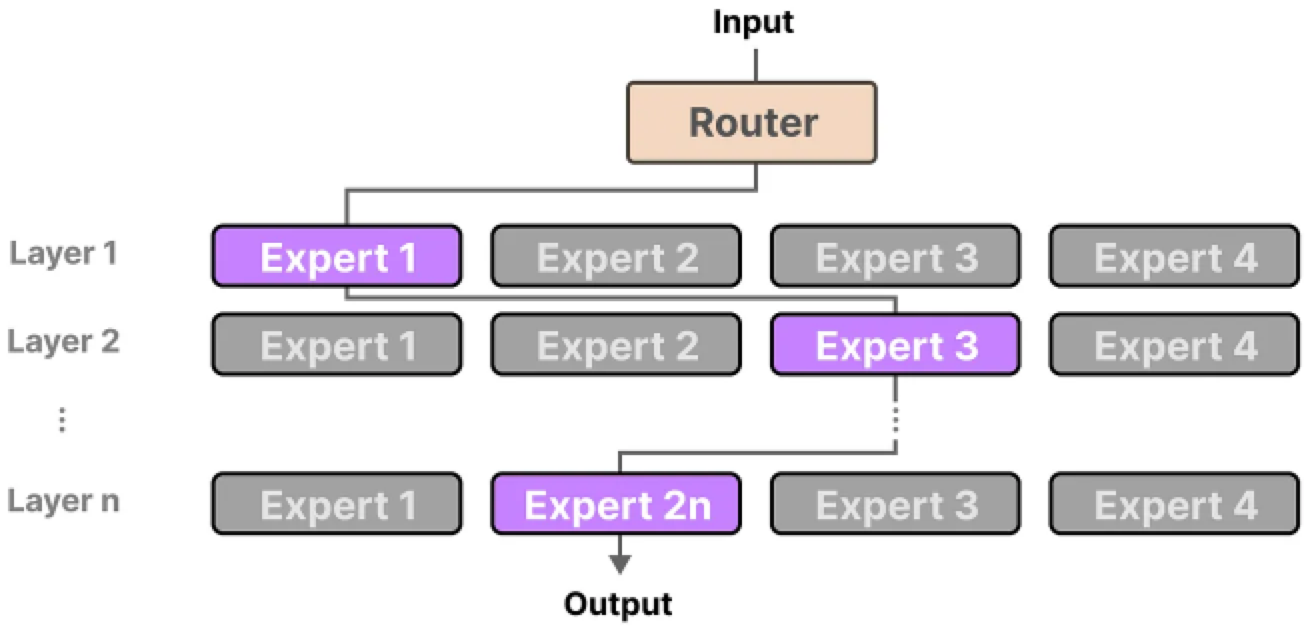
- MoE is supposed to replace; the *dense layers*.
- Each layer is composed of 8 feed forward blocks known as experts

https://www.youtube.com/watch?v=sOPDGQjFcuM
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts

In each layer of an LLM with an MoE, we find (somewhat specialized) experts:

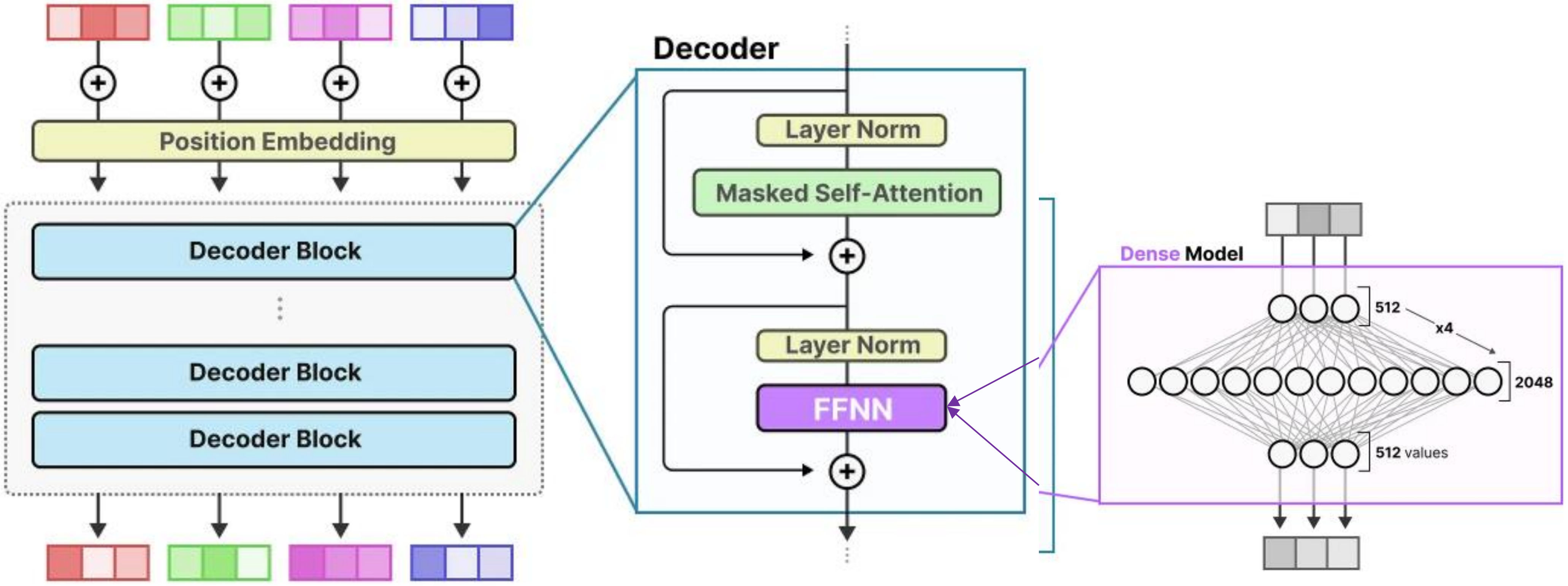

Know that an “expert” is not specialized in a specific domain like “Psychology” or “Biology”. At most, it learns syntactic information on a word level instead:

The *router* (gate network) selects the expert(s) best suited for a given input:




Each expert is not an entire LLM but a submodel part of an LLM's architecture.

# Recall Transformers Arch & FFNN



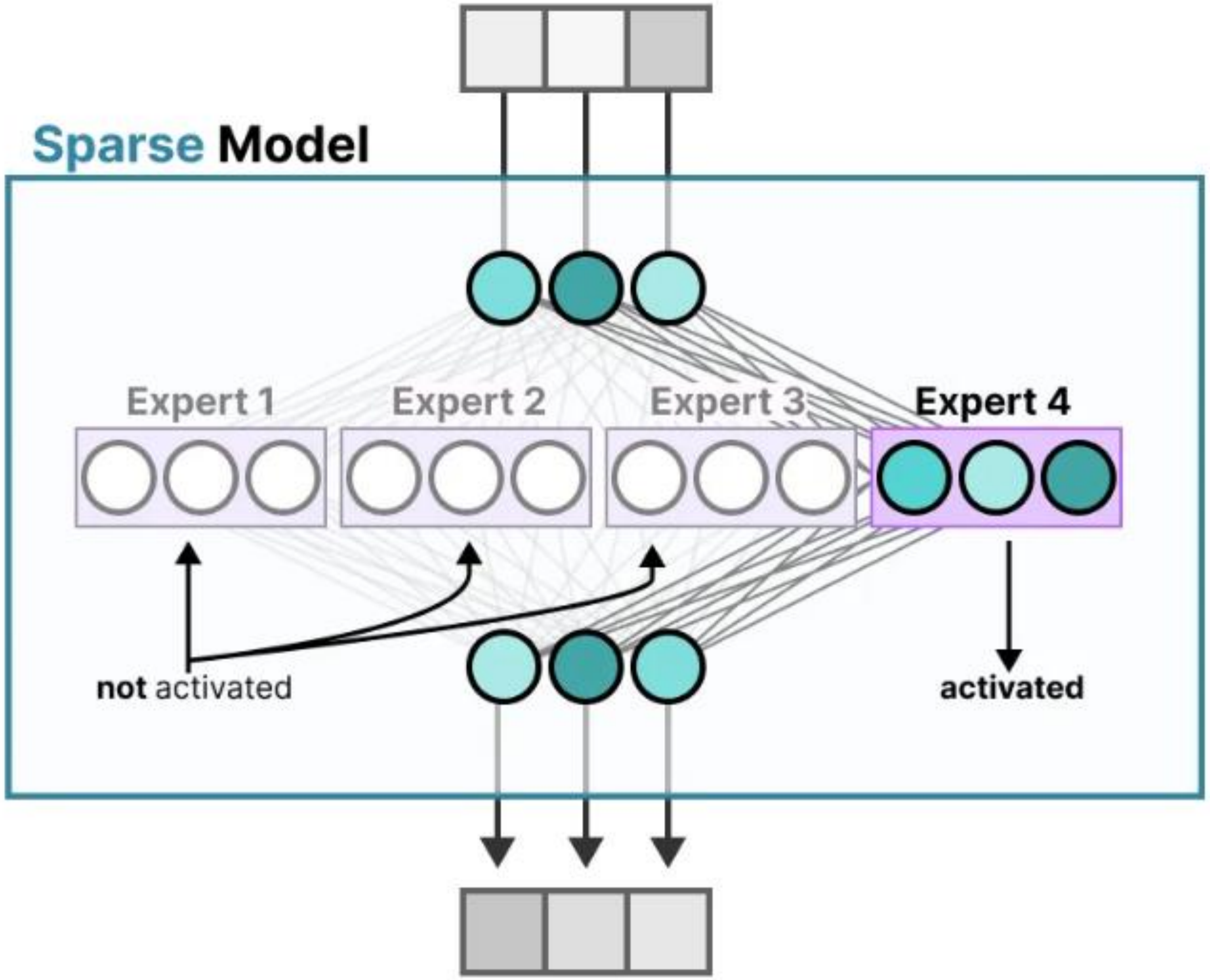

The FFNN in a traditional Transformer is called a **dense** model since all parameters (its weights and biases) are activated.

# Sparse Models

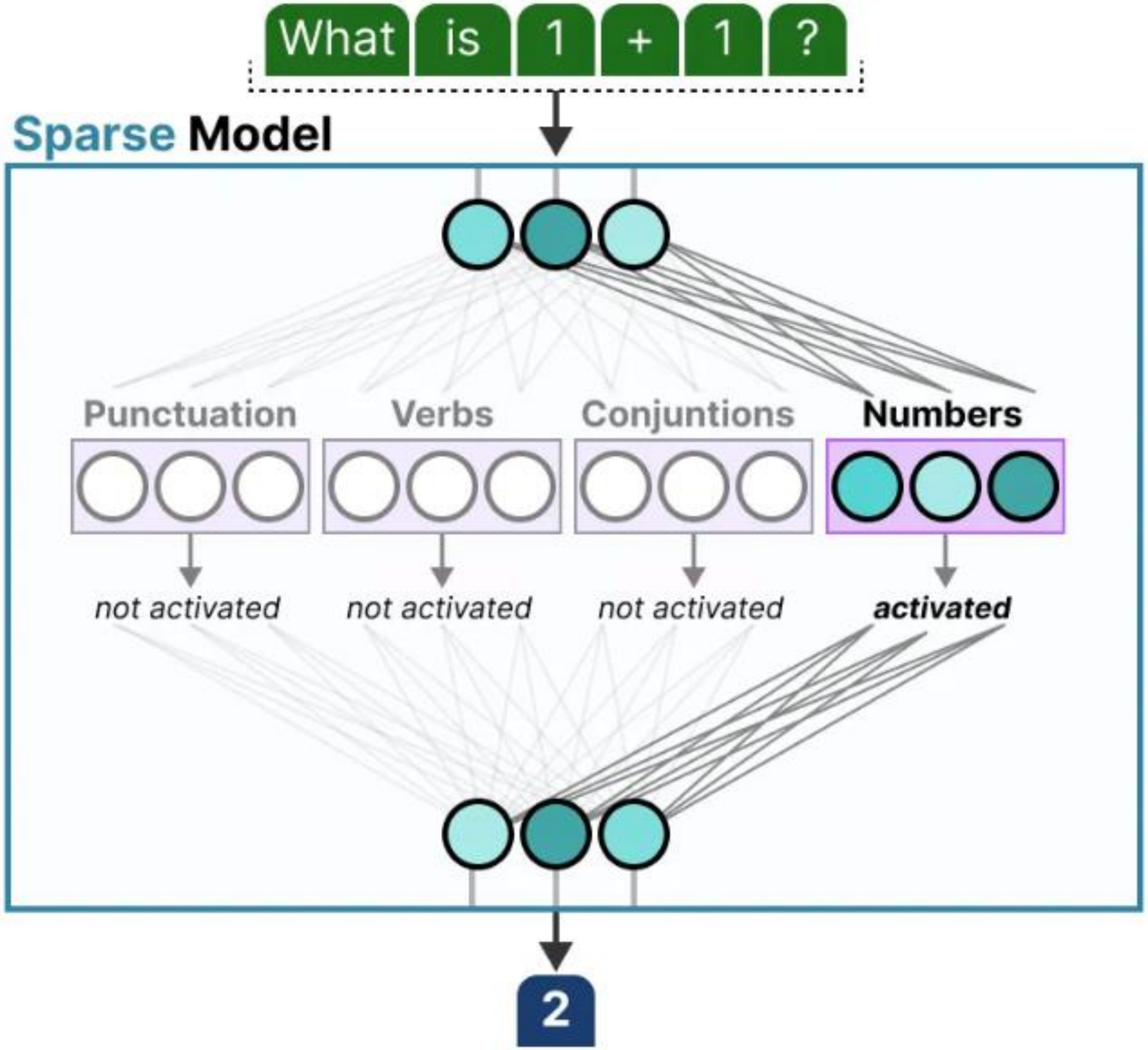
- In contrast, **sparse** models only activate a portion of their total parameters and are closely related to Mixture of Experts.

- To illustrate, we can chop up our dense model into pieces (so-called experts), retrain it, and only activate a subset of experts at a given time.

- The underlying idea is that each expert learns different information during training.
- Then, when running inference, only specific experts are used as they are most relevant for a given task.



When asked a question, we can select the expert best suited for a given task.
experts learn more fine-grained information than entire domains

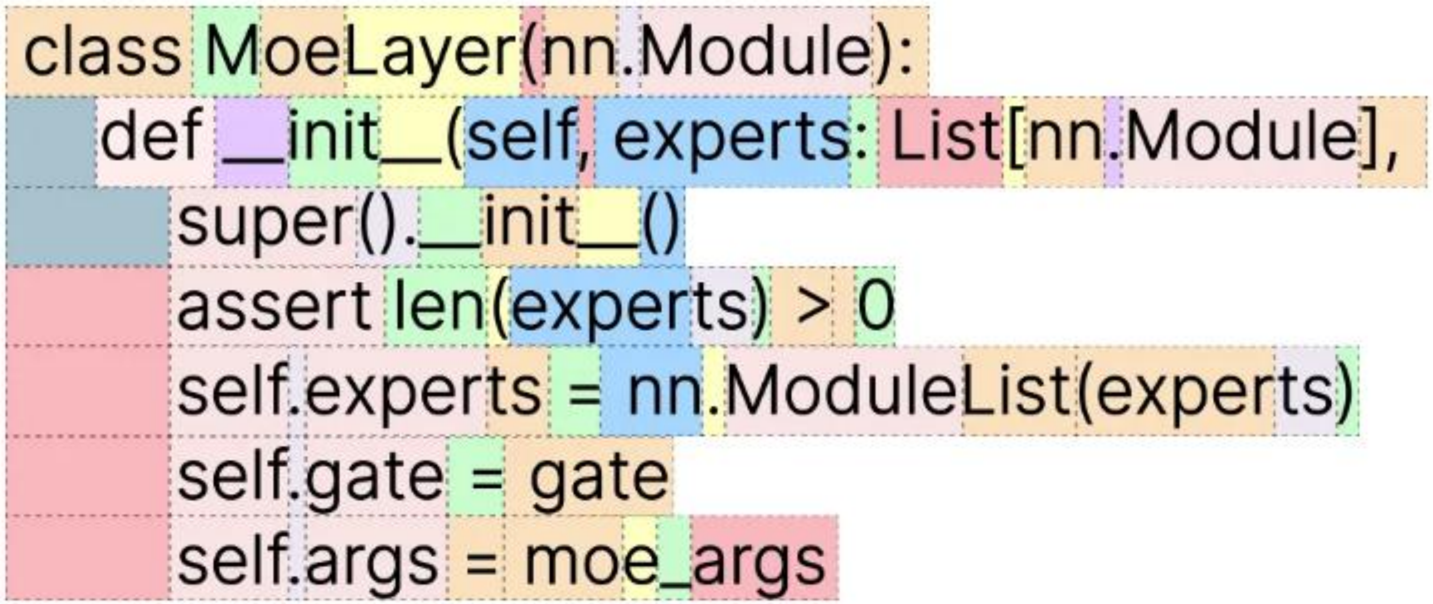


| Expert specialization | Expert position | Routed tokens |
|---|---|---|
| **Punctuation** | Layer 2 | , , , , , , , , , - , , , , , ). ) |
| | Layer 6 | , , , , , : . : , & , & & ? &- ,, ? ,,, . |
| **Conjunctions and articles** | Layer 3 | The the the the the the the the The... |
| | Layer 6 | a and and and and and and or and ... |
| **Verbs** | Layer 1 | died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died... |
| **Visual descriptions** *color, spatial position* | Layer 0 | her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue |
| **Counting and numbers** *written and numerical forms* | Layer 1 | after 37 19. 6. 27 I I Seven 25 4, 54 I two dead we Some 2012 who we few lower |

Expert specialization of an encoder model in the ST-MoE paper.

Experts in decoder models, however, do not seem to have the same type of specialization. That does not mean though that all experts are equal.
A great example can be found in the Mixtral 8x7B paper where each token is colored with the first expert choice.

A great example can be found in the [Mixtral 8x7B paper]() where each token is colored with the first expert choice.

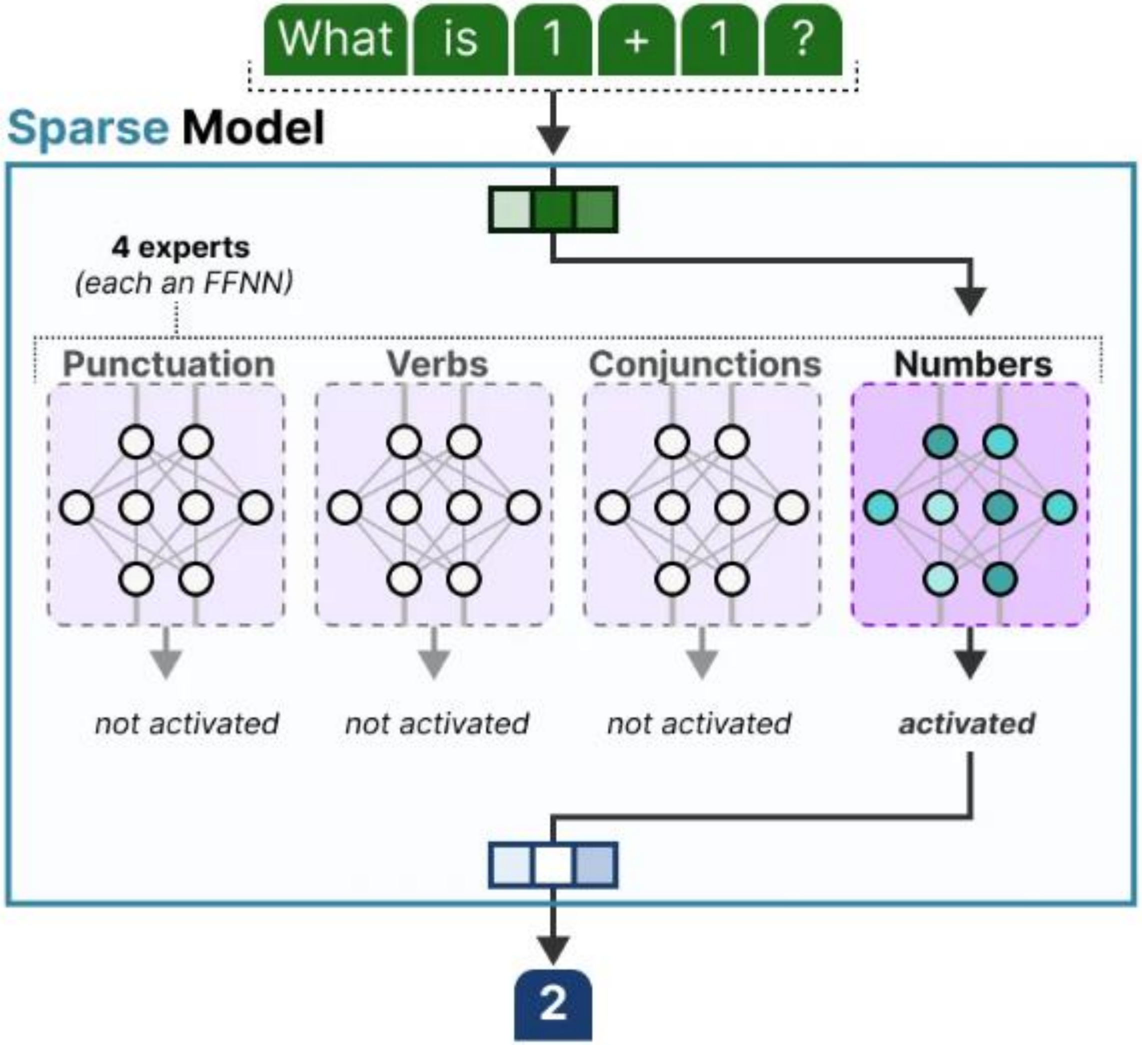



- Experts don’t focus on specific domains (e.g., NLP, math, code) but rather on token types (e.g., function names, variables, operators).

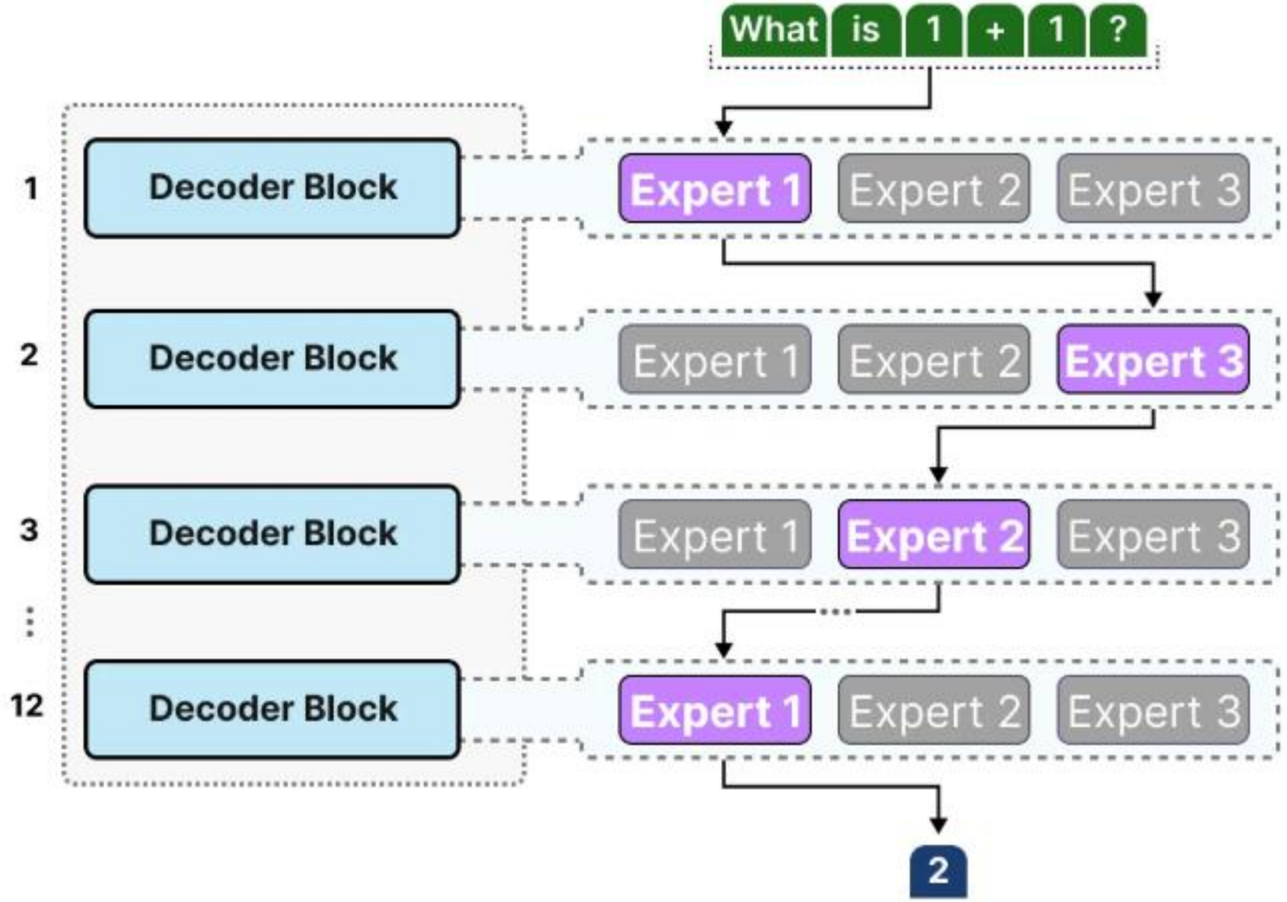
- This visual also demonstrates that experts tend to focus on syntax rather than a specific domain.

- Thus, although decoder experts do not seem to have a specialism they do seem to be used consistently for certain types of tokens.

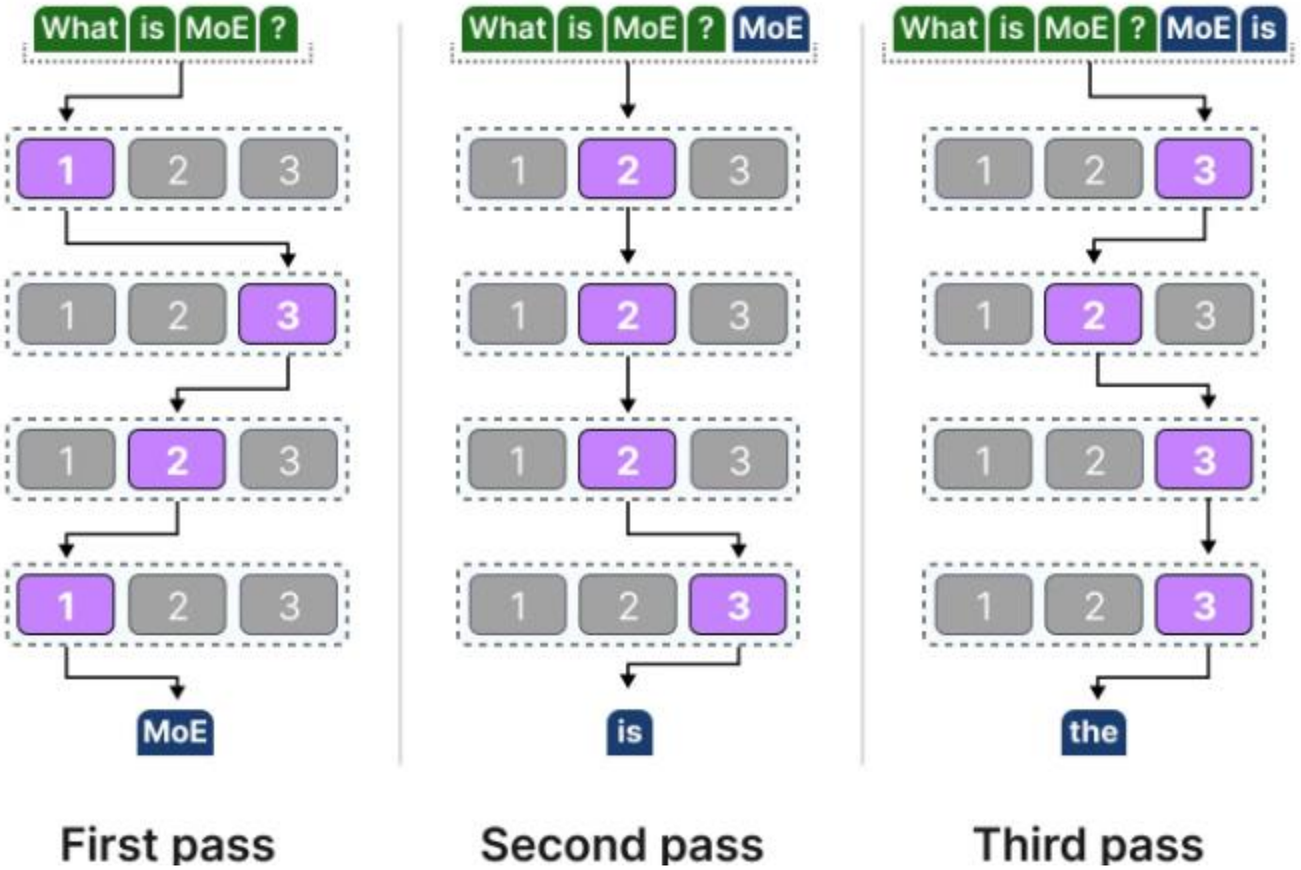
# MoE Architecture



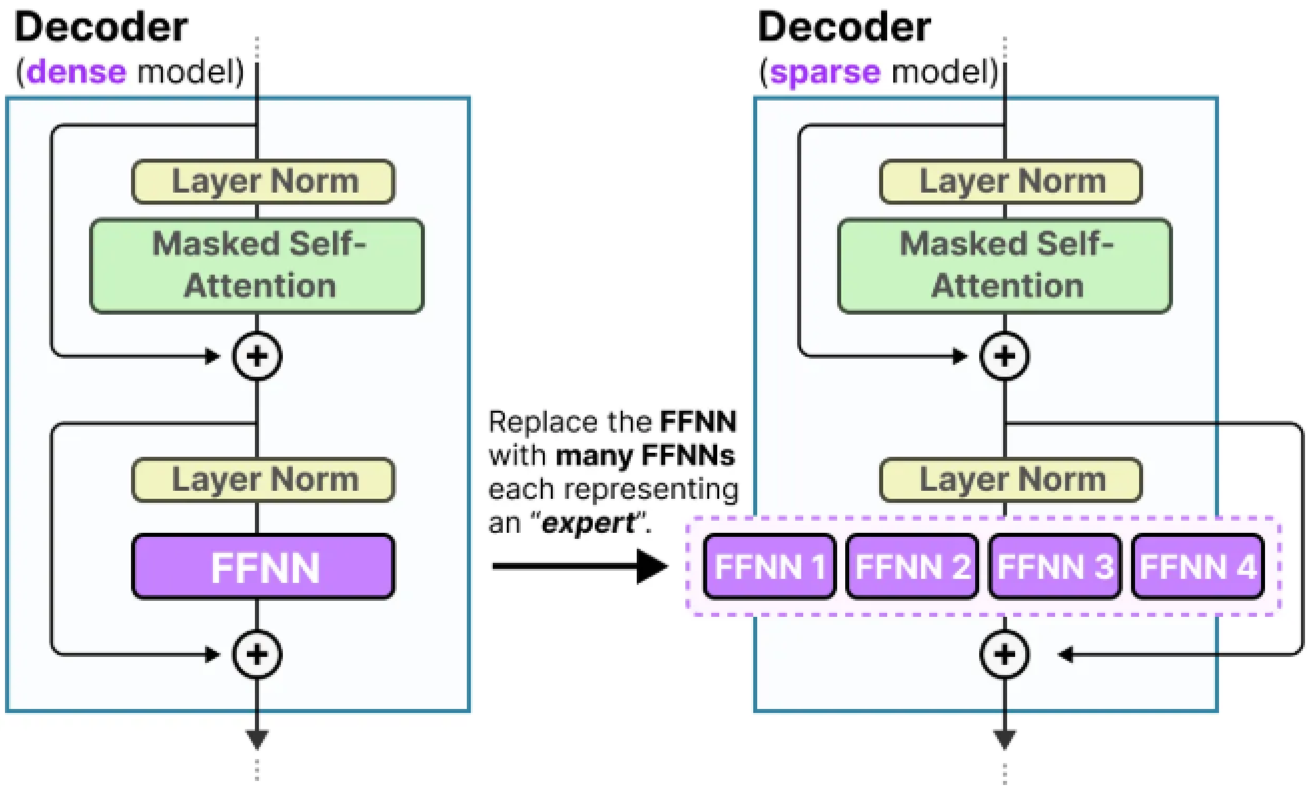

Since most LLMs have several decoder blocks, a given text will pass through multiple experts before the text is generated:



The chosen experts likely differ between tokens which results in different "paths" being taken:



If we update our visualization of the decoder block, it would now contain more FFNNs (one for each expert) instead:




The decoder block now has multiple FFNNs (each an "expert") that it can use during inference.

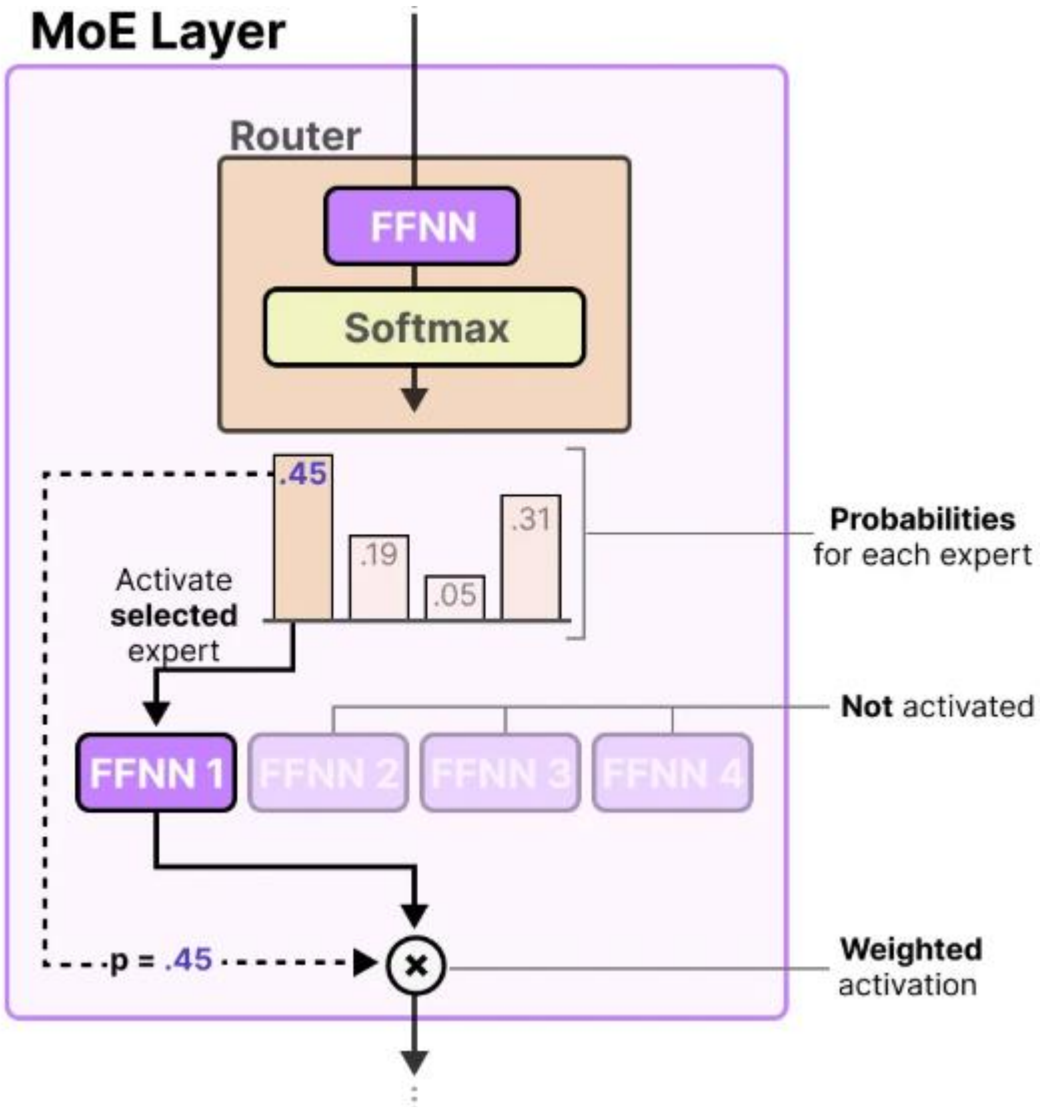
# The Routing Mechanism

Now that we have a set of experts, how does the model know which experts to use?

Just before the experts, a **router** (also called a **gate network**) is added which is trained to choose which expert to choose for a given token.

- The **router** (or **gate network**) is also an FFNN and is used to **choose the expert** based on a particular **input**.
- It outputs probabilities which it uses to select the best matching expert.

The expert layer returns the **output** of the selected expert **multiplied** by the **gate value** (selection probabilities).



# Mistral 8x7B : the gating/routing function

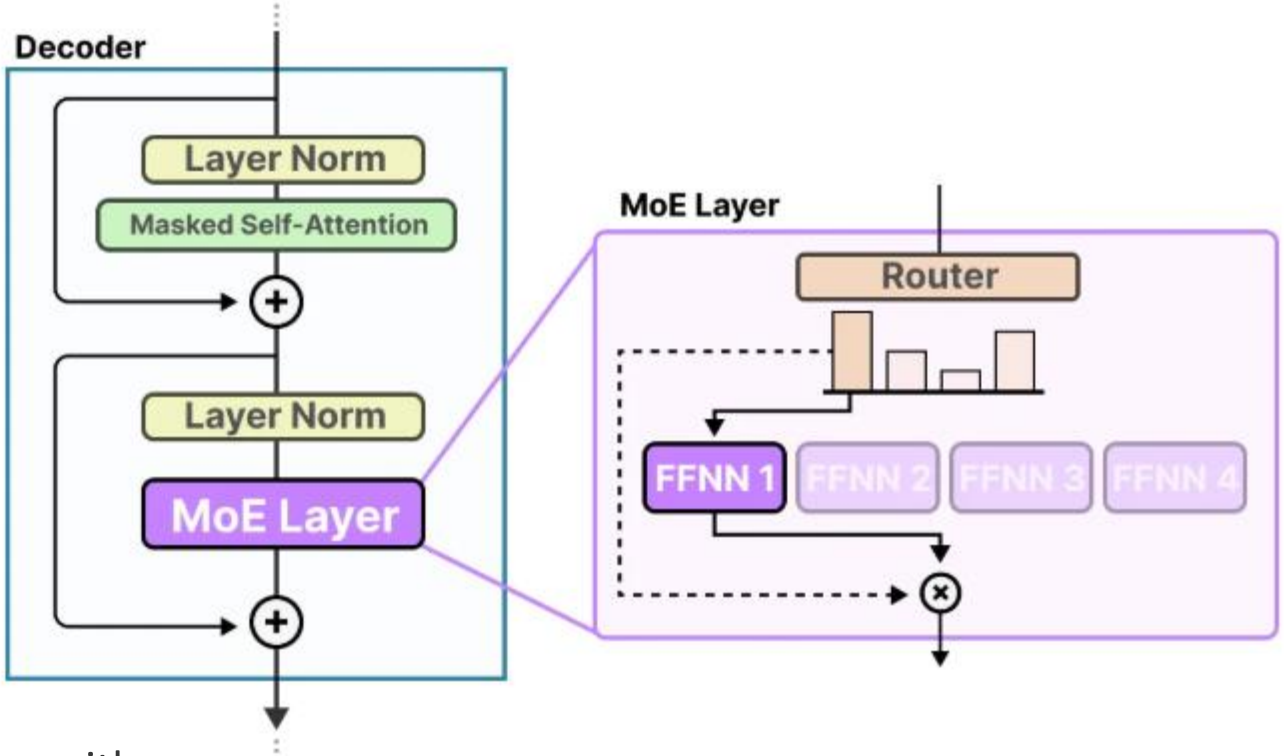
- The gating function is just a linear layer **(in_features=4096, out_feature=8, bias=False)** that's trained along with the rest of the model.
  For each token embedding, it produces 8 logits, which indicate which expert to select.

**Mistral 8x7B**

**Mistral 7B**

```
self.feed_forward: nn.Module
if args.moe is not None:
    self.feed_forward = MoeLayer(
        experts=[FeedForward(args=args) for _ in range(args.moe.num_experts)],
        gate=nn.Linear(args.dim, args.moe.num_experts, bias=False),
        moe_args=args.moe,
    )
else:
    self.feed_forward = FeedForward(args=args)
```

- The expert layer returns the output of the selected expert multiplied by the gate value (selection probabilities).
- The router together with the experts (of which only a few are selected) makes up the **MoE Layer**:

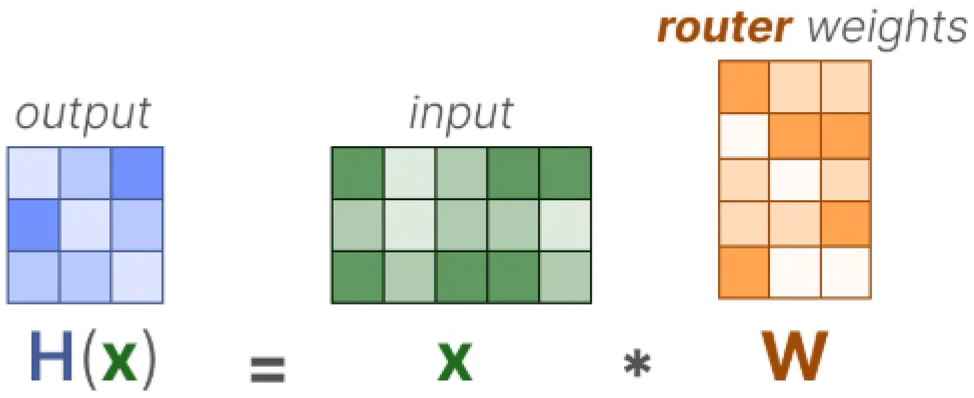



- A given MoE layer comes in two sizes, either a *sparse* or a ***dense*** mixture of experts.
- Both use a router to select experts but a **Sparse MoE only selects a few** whereas a **Dense MoE selects them all** but potentially in different distributions.

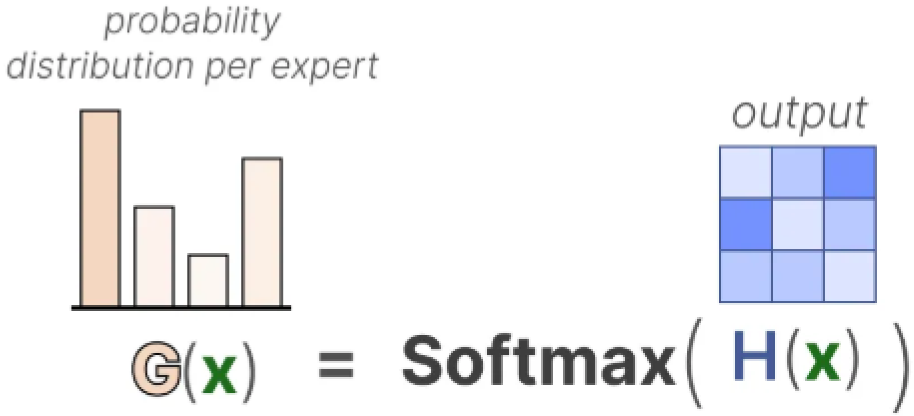
# Selection of Experts

The gating network is arguably the most important component of any MoE as it not only decides which experts to choose during *inference* but also *training*.

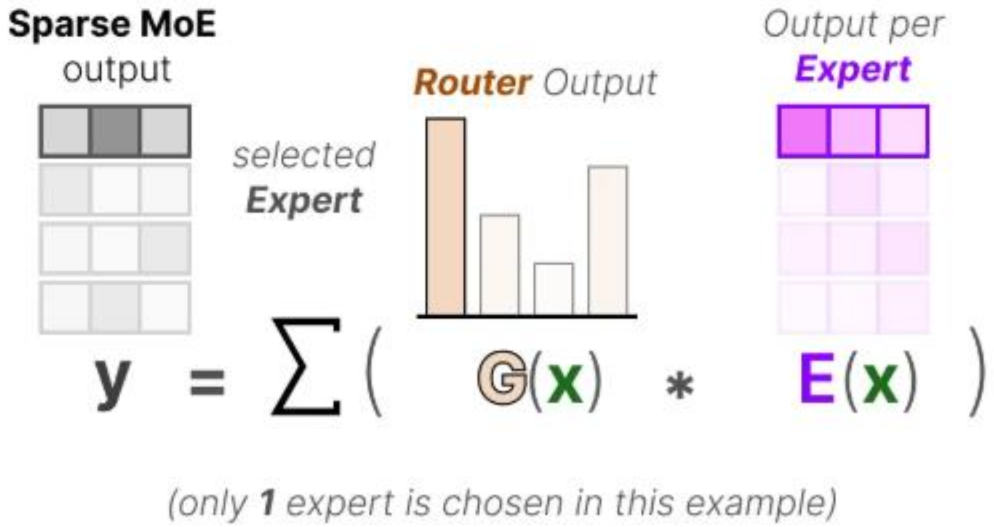
In its most basic form, we multiply the input ($x$) by the router weight matrix ($W$):



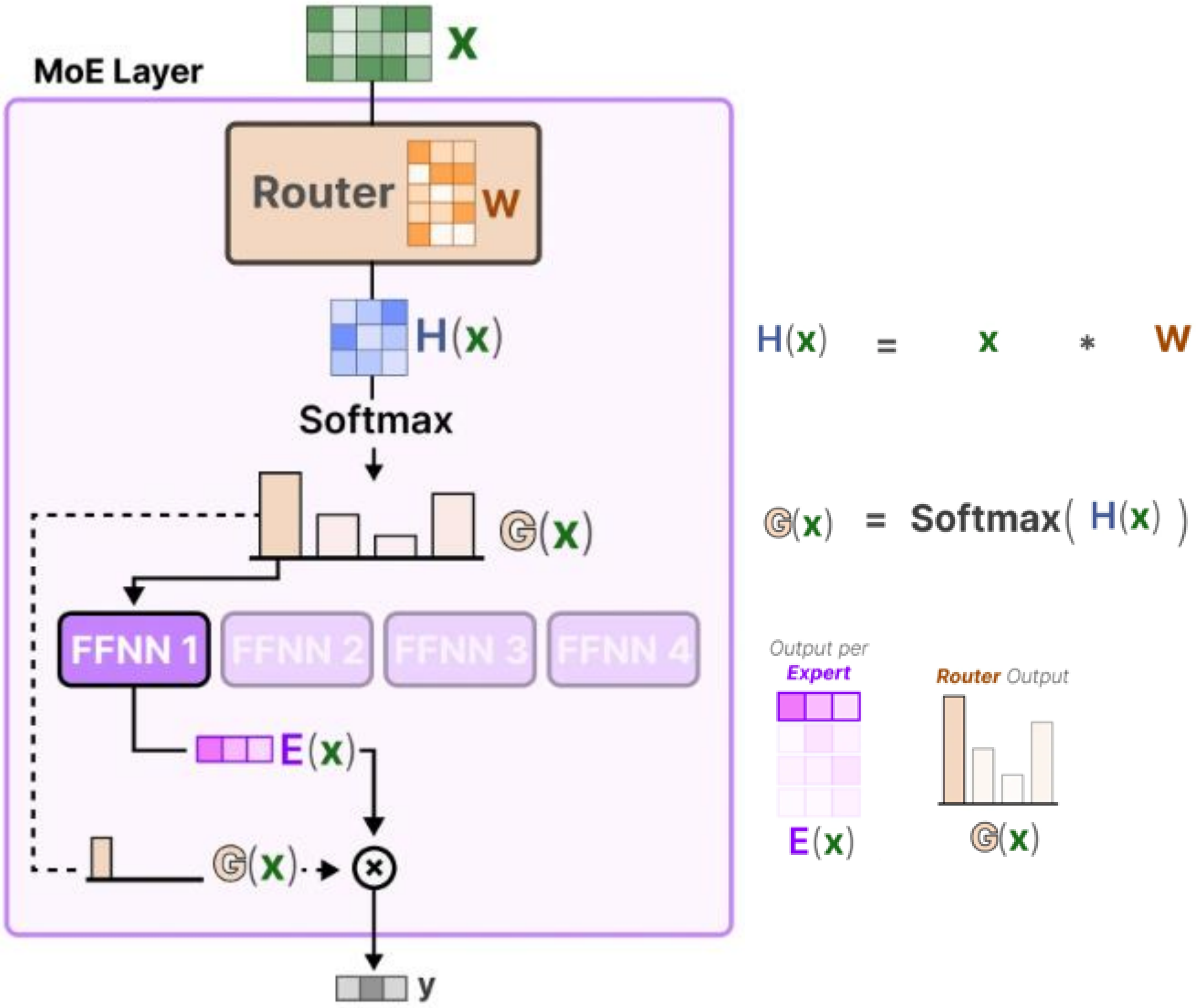

Then, we apply a **SoftMax** on the output to create a probability distribution G($x$) per expert:




The router uses this probability distribution to choose the **best matching expert** for a given input.



Let’s put everything together and explore how the input flows through the router and experts:



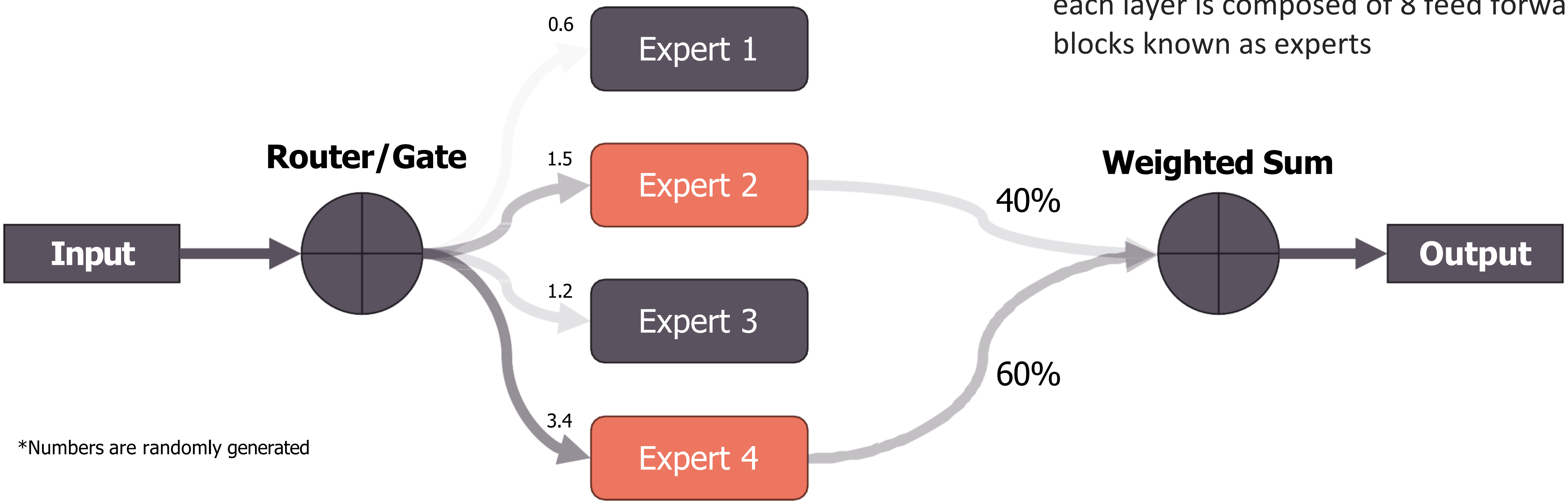
# Mixture of Experts: an introduction

Mixture of Experts is an ensemble technique, in which we have multiple "expert" models, each trained on a subset of the data, such that each model specializes on it and then the output of the experts are combined (usually a weighted sum or by averaging) to produce one single output.

In the case of **Mistral 8x7B**, we talk about **Sparse Mixture of Experts (SMoE)**, because only 2 out of 8 experts are used for every token.

The gate produces **logits** which are used to select the top-k experts. The top-k logits are then run through a **softmax** to produce weights.

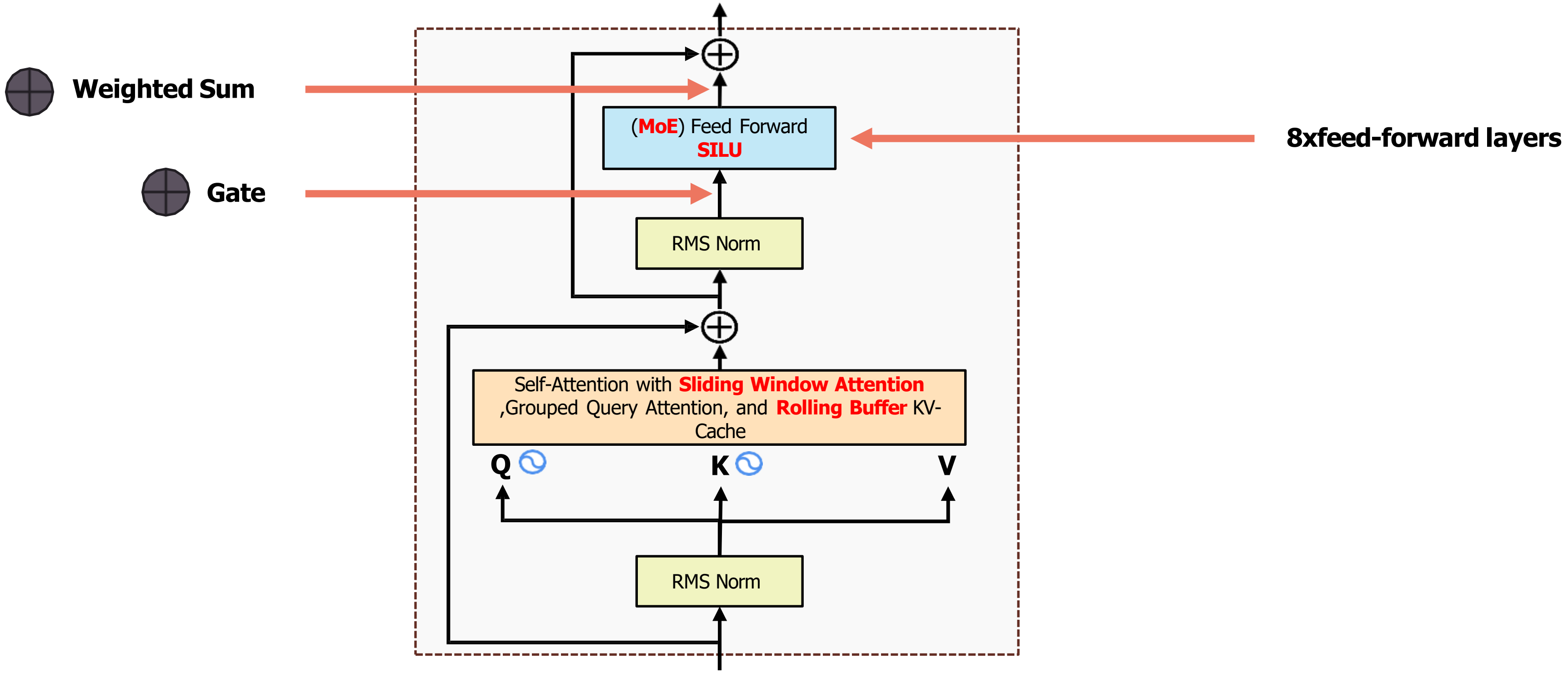
each layer is composed of 8 feed forward blocks known as experts




*Numbers are randomly generated

# Mistral 8x7B: expert feed-forward layers

- In the case of Mistral 8x7B, the experts are the Feed-Forward layers present at every Encoder layer. Each Encoder layer is comprised of a single Self-Attention mechanism, followed by a mixture of experts of 8 FFN. The gate function selects the top 2 experts for each incoming token. The output is combined with a weighted sum.
- This allows to increase the parameters of the model, but without impacting the computation time, since the input will only pass through the top 2 experts, so the intermediate matrix multiplications will be performed only on the selected experts.



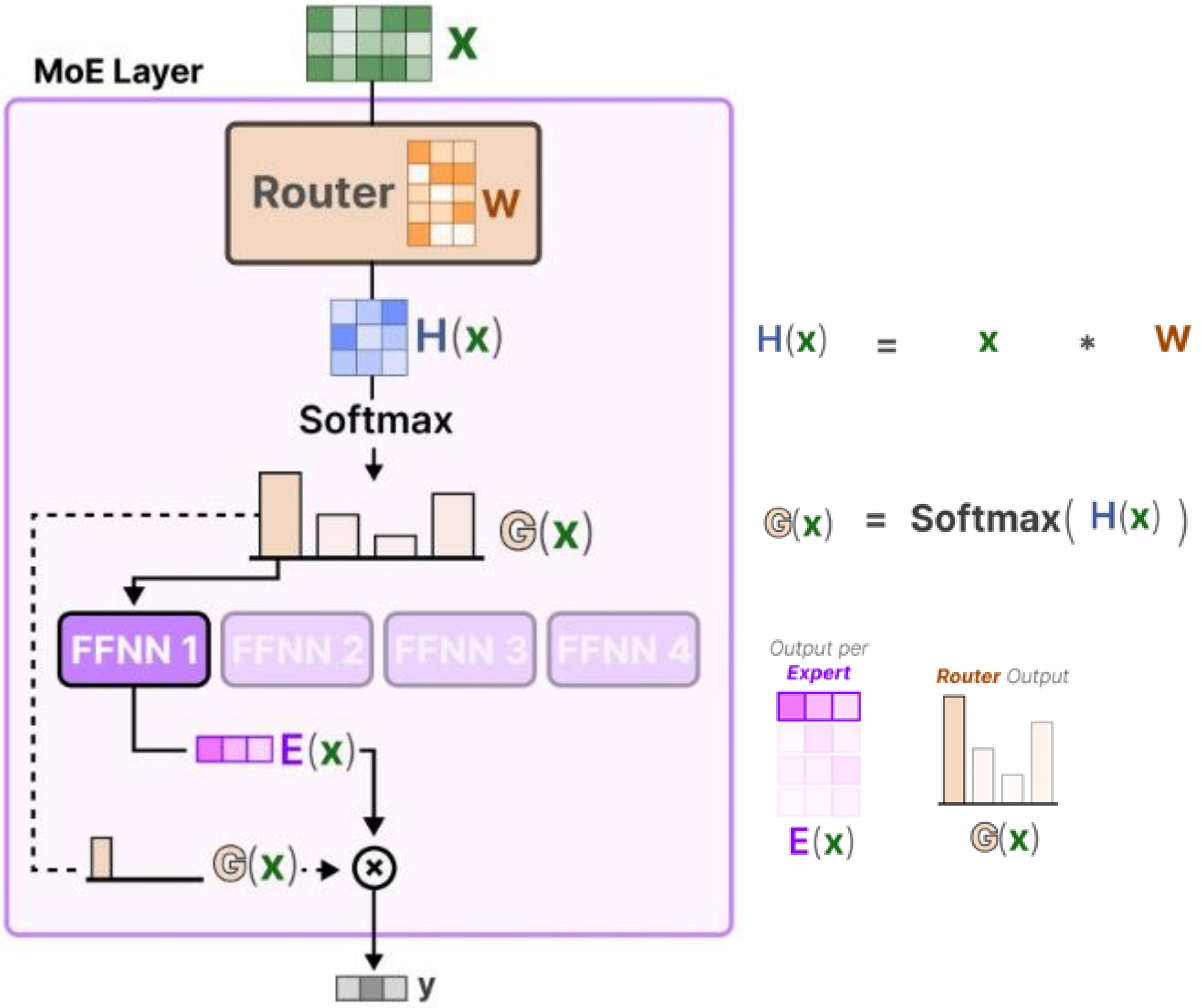
# Mixture of Experts: why we apply the softmax after selecting the top-k experts?

- If we apply the softmax directly to the output of the gate function, **this will result in a “probability distribution” (weights that sum up to 1) over all the experts.**
- But since we are only going to use the top-k of them, we want a “probability distribution” over only the selected experts.
- This also makes it easier to compare two models trained on different number of experts, since the sum of the weights applied to the output will always be 1 independently on the number of experts chosen by the gate function.

- Applying softmax after selection has some issues:
- If an expert receives a very small probability due to normalization over all experts, it might still get selected, even though it isn't the best candidate.
- Also, models trained with **different numbers of experts** would behave differently because the softmax probabilities would be normalized over different ranges.

Let's put everything together and explore how the input flows through the router and experts:

**Complexity of Routing:**
What if the same experts are chosen over others?



Let’s put everything together and explore how the input flows through the router and experts:

## Complexity of Routing:

What if the same experts are chosen over others?

- Not only will there be an **uneven distribution of experts** chosen, but some experts will hardly be trained at all.
- This results in issues during both training and inference.
- Instead, we want **equal importance** among experts during training and inference, which we call **load balancing**. In a way, it’s to prevent overfitting on the same experts.