EdgeAI-Mixture of Experts (MoE) on Raspberry Pi

From mathematical foundations to edge implementation

Social media:

👨🏾‍💻 Github: thommaskevin/TinyML

👷🏾 Linkedin: Thommas Kevin

📽 Youtube: Thommas Kevin

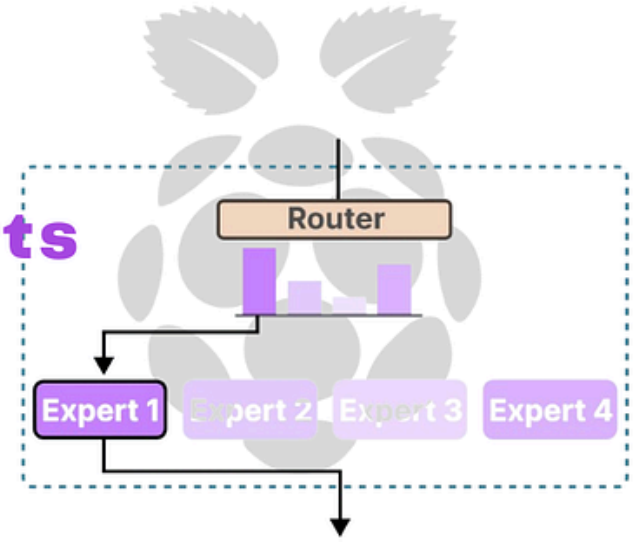
:pencil2:CV Lattes CNPq: Thommas Kevin Sales Flores

👨🏾‍🏫 Research group: Conecta.ai

EdgeAI

Mixture of Experts on Raspberry Pi

"Comprehensive guide to deploying on edge devices."



SUMMARY

---

1 - Introduction

The **Mixture-of-Experts (MoE)** paradigm is a machine learning technique that has gained considerable prominence, particularly in the context of **Large Language Models (LLMs)**. Its main appeal lies in the ability to **scale model capacity** without incurring a proportional increase in computational cost. At its core, an MoE employs multiple neural sub-networks, referred to as *experts*, to partition the problem space into more homogeneous regions. This design embodies a form of ensemble learning, where specialized components collaborate to enhance overall performance.

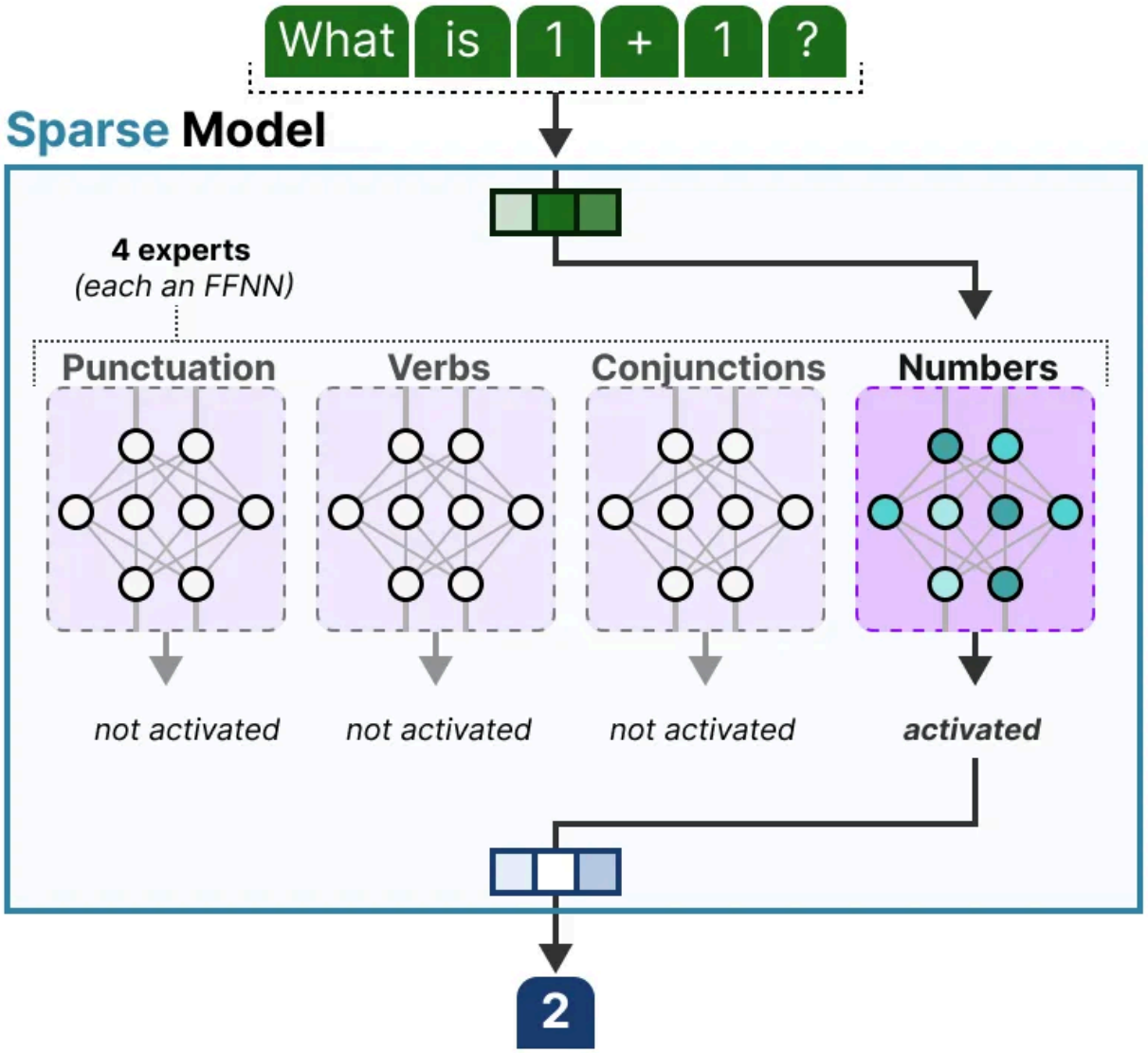
1.1 - Why MoE?

The motivation for adopting MoE architectures stems from several key advantages:

- **Model Capacity** MoE allows for an increase in model capacity—the degree of complexity that the model can capture—without requiring all parameters to be active during inference. This is achieved by replacing dense layers with MoE layers, where each expert sub-network has the same dimensionality as the original dense layer [3].
- **Computational Efficiency** Sparse MoE models exhibit improved efficiency in terms of FLOPs (floating-point operations) per parameter. Instead of activating all parameters, only a selected subset of experts is engaged for each input. This selective activation reduces the computational cost per token [3].
- **Reduced Training Cost** Because FLOPs per weight are used more efficiently, MoE enables the training of larger and more capable models under fixed computational or financial constraints. In practice, this means that more tokens can be processed within the same budget [3].
- **Lower Latency** In scenarios where computational resources form a bottleneck, MoE can reduce the latency associated with generating the first token. This property is particularly valuable in applications that demand multiple sequential calls to the model.

1.2-Components of MoE

The architecture of a Mixture-of-Experts can be understood through two fundamental components:



1.2.1 - Experts

Experts are neural sub-networks—most commonly **Feedforward Neural Networks (FFNNs)**—that collectively form the mixture. Within an MoE layer, each traditional dense FFNN is replaced by a pool of experts, of which only a subset is activated for a given input. Each expert receives the same input and produces an output vector [1].

It is important to emphasize that experts are **not** domain-specific (e.g., “Psychology” or “Biology”). Instead, their specialization emerges at a **syntactic or token level**, allowing them to handle particular linguistic or contextual patterns [4].

1.2.2 - Router (Gating Network)

The Router—also known as the **Gating Network**—is responsible for selecting which experts should be activated for a given input. It processes the input and generates a weight vector, where each weight reflects the relative importance of an expert [1].

During training, the router learns to route tokens toward the most relevant experts, thereby ensuring both efficiency and specialization [4].

Here is your text rewritten in **formal English** with clear **subsections by MoE components**, improving structure and readability while keeping all the mathematical rigor intact:

2—Mathematical Foundations

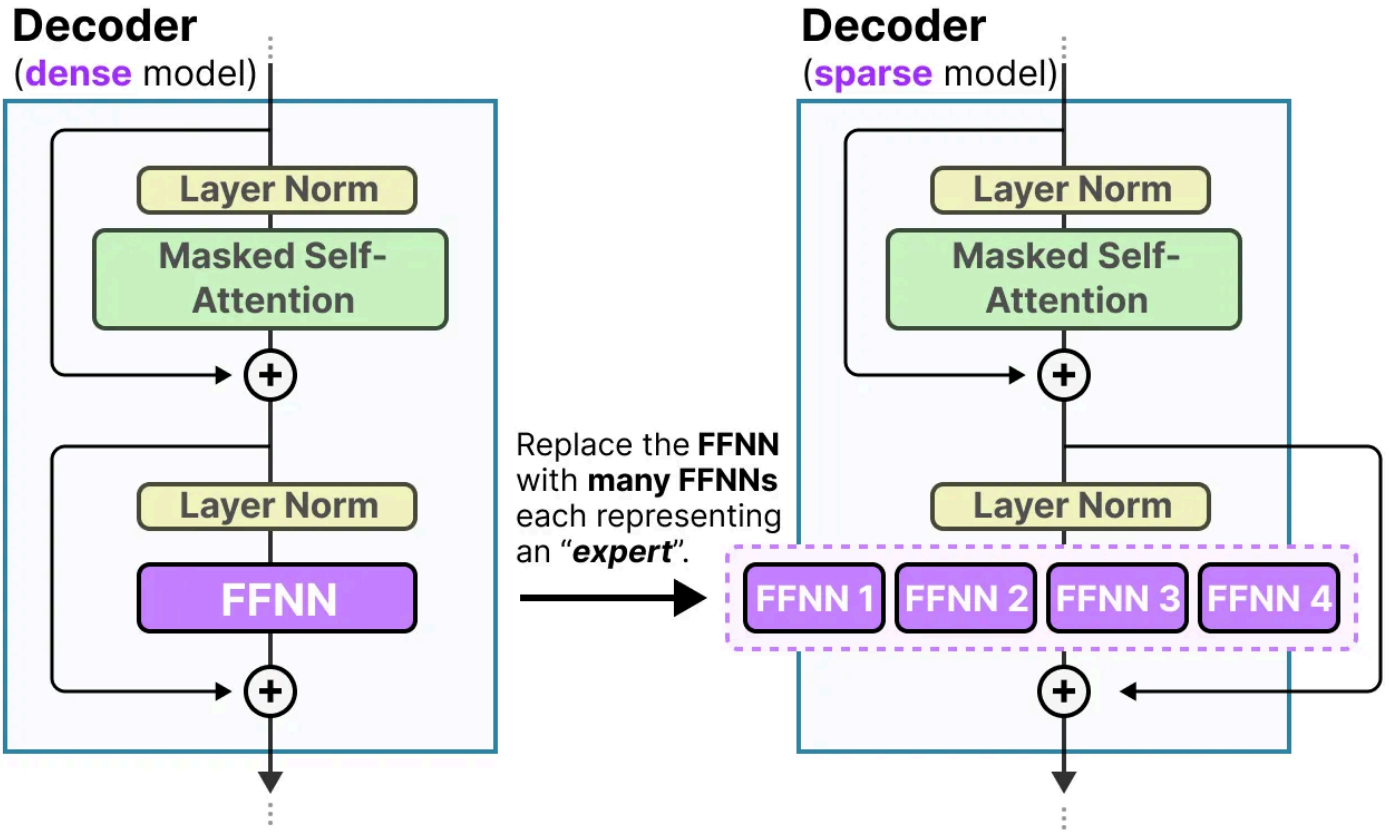
A **Mixture-of-Experts (MoE)** model partitions the mapping from inputs to outputs across a set of specialized submodels, known as **experts**, and employs a **gating network** to determine how expert outputs are combined for each input. This section presents the mathematical formulation and training dynamics of MoE, organized by its key components.

2.1—Experts

Let $x \in \mathrm{R}^d$ denote an input vector, and let M denote the number of experts. Each expert i is parameterized by θ_i and defines a function

$$f_i(x; \theta_i).$$

Depending on the task, an expert may output a scalar value, a vector, or a probability distribution. The experts collectively define the representational capacity of the MoE model.



2.2—Gating Network (Router)

The gating network or Router, parameterized by ϕ, determines the contribution of each expert to the final prediction. It outputs a vector of non-negative weights:

$$g(x;\phi) = (g_1(x;\phi), \dots, g_M(x;\phi)),$$

with the constraints

$$g_i(x;\phi) \geq 0, \quad \sum_{i=1}^{M} g_i(x;\phi) = 1.$$

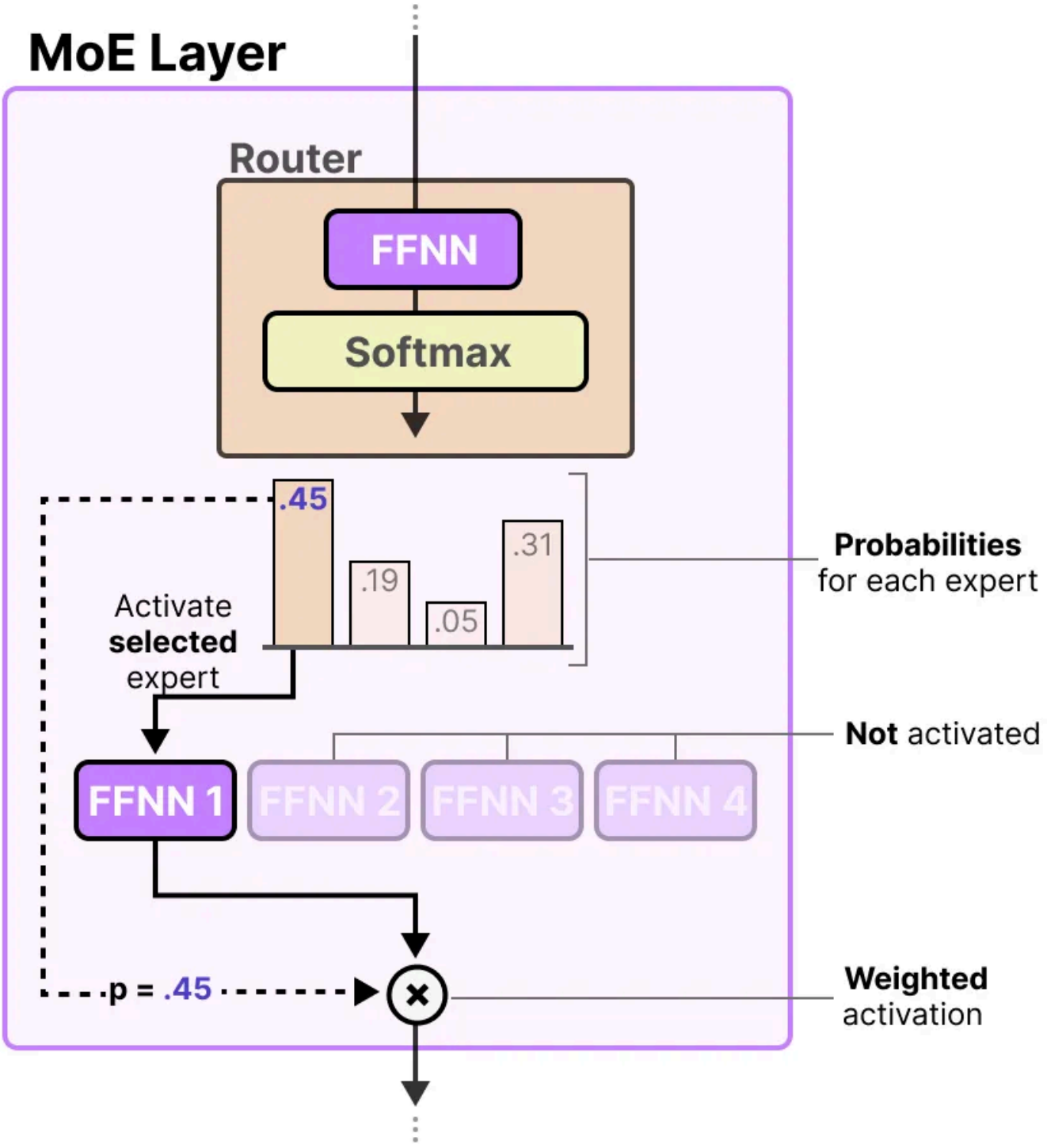
Thus, the prediction is the convex combination

$$\hat{y}(x) = \sum_{i=1}^{M} g_i(x;\phi) f_i(x;\theta_i).$$

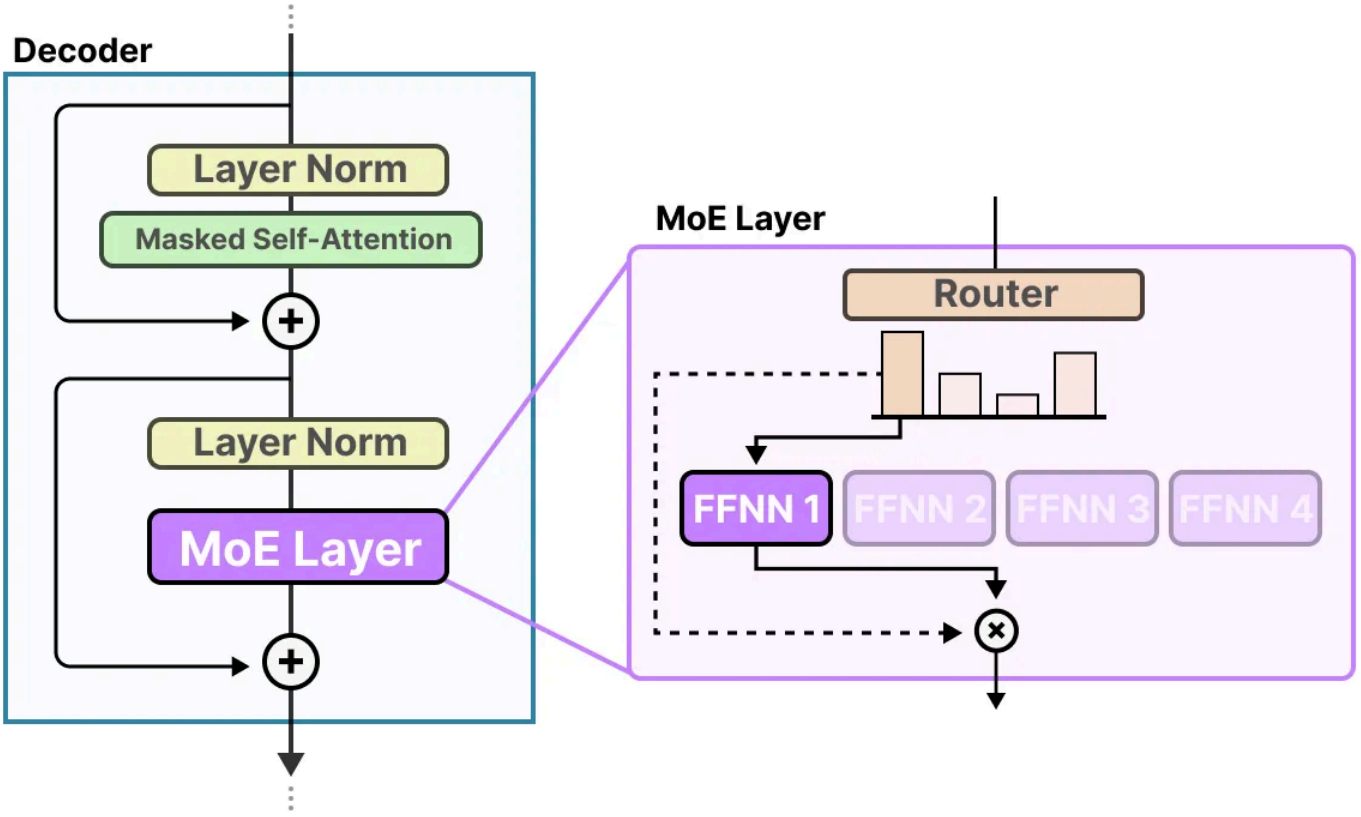
A common formulation computes **gating logits** $h_i(x;\phi)$ (e.g., $h_i(x;\phi) = w_i^\top x + b_i$ or via a small neural network) and applies the softmax:

$$g_i(x;\phi) = \frac{\exp(h_i(x;\phi))}{\sum_{j=1}^{M} \exp(h_j(x;\phi))}.$$

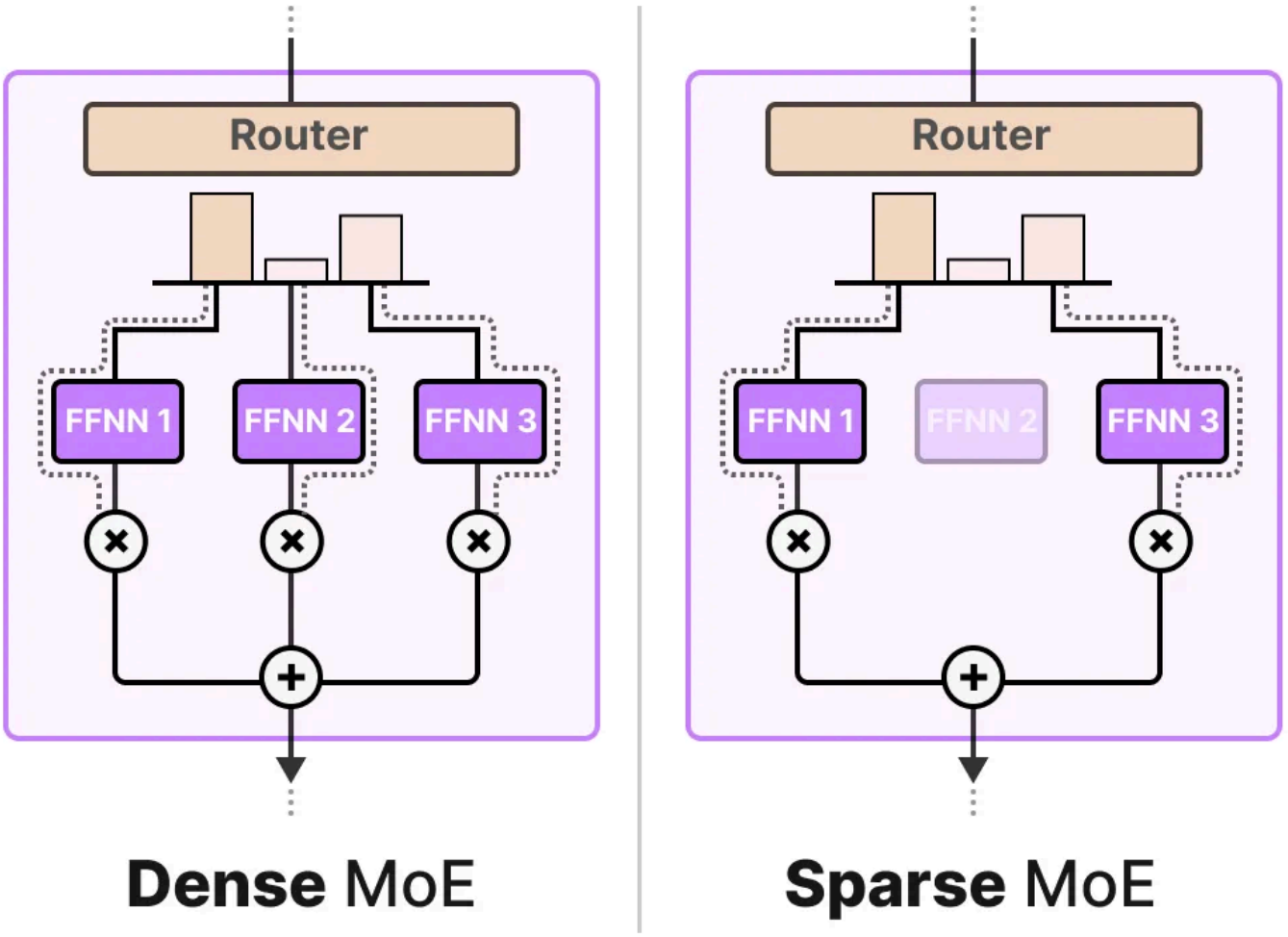
The Router outputs probabilities which it uses to select the best matching expert:



The MoE layer is composed of the router and a subset of experts, with only a limited number being activated for each input:



An MoE layer can be organized in two main ways: sparse or dense. In both cases, the router is responsible for directing the input to the experts. In the Sparse MoE, only a few experts are chosen to process the information, which makes the computation more efficient. In contrast, the Dense MoE activates all experts, but their contributions may differ depending on the distribution of weights. Figure X below illustrates this distinction between the two configurations.



2.3—Training Objectives

For supervised learning with dataset $(x^{(n)}, y^{(n)})_{n=1}^{N}$, the MoE is trained to minimize an empirical risk.

- **Regression (squared error):**

$$\ell^{(n)} = \tfrac{1}{2}\|y^{(n)} - \hat{y}(x^{(n)})\|^2, \quad \mathrm{L} = \tfrac{1}{N}\sum_{n=1}^{N} \ell^{(n)}.$$

- **Classification (cross-entropy):**

$$\mathrm{L} = -\tfrac{1}{N}\sum_{n=1}^{N}\sum_{c} y_c^{(n)} \log \hat{y}_c^{(n)},$$

with

$$\hat{y}^{(n)} = \sum_{i=1}^{M} g_i(x^{(n)};\phi) f_i(x^{(n)};\theta_i).$$

2.4—Gradient Flow

Because the MoE output depends jointly on experts and the gating network, gradients propagate through both.

- **Gradients for experts:**

$$\frac{\partial \ell}{\partial \theta_i} = g_i(x;\phi)\,\frac{\partial \ell}{\partial f_i(x;\theta_i)}\,\frac{\partial f_i(x;\theta_i)}{\partial \theta_i}.$$

Thus, expert i receives gradient contributions weighted by $g_i(x;\phi)$.

- **Gradients for gating parameters:**

Let $u_i = g_i(x;\phi)$ and $h_i = h_i(x;\phi)$. Since

$$\frac{\partial u_i}{\partial h_k} = u_i(\delta_{ik} - u_k),$$

the chain rule gives

$$\frac{\partial \ell}{\partial h_k} = \sum_{i=1}^{M} \langle \tfrac{\partial \ell}{\partial \hat{y}}, f_i \rangle\, u_i(\delta_{ik} - u_k).$$

Finally, $\partial \ell / \partial \phi$ follows by backpropagation through $h_k(x;\phi)$.

2.5—Probabilistic Interpretation

Introducing a latent variable $z \in 1, \ldots, M$ denoting the chosen expert, the conditional likelihood is a **mixture distribution**:

$$p(y \mid x) = \sum_{i=1}^{M} p(z = i \mid x)\, p(y \mid x, z = i),$$

where $p(z = i \mid x) = g_i(x;\phi)$.

The log-likelihood over the dataset is

$$\sum_{n} \log \Big(\sum_{i=1}^{M} g_i(x^{(n)};\phi)\, p(y^{(n)} \mid x^{(n)}, z = i;\theta_i) \Big).$$

Expectation–Maximization (EM) provides an alternative training view:

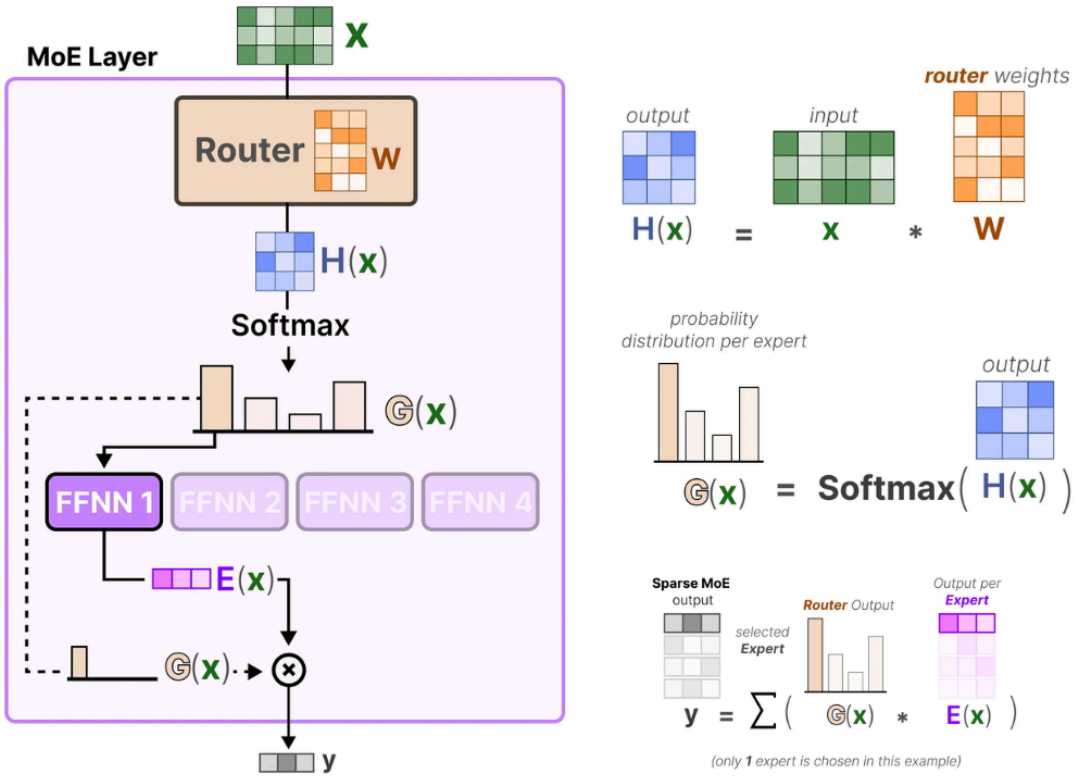
- **E-step (responsibilities):**

$$r_{n,i} = \frac{g_i(x^{(n)};\phi)\, p(y^{(n)} \mid x^{(n)}, z=i;\theta_i)}{\sum_j g_j(x^{(n)};\phi)\, p(y^{(n)} \mid x^{(n)}, z=j;\theta_j)}.$$

- **M-step (maximization):**

$$\sum_{n,i} r_{n,i}\big(\log g_i(x^{(n)};\phi) + \log p(y^{(n)} \mid x^{(n)}, z=i;\theta_i)\big).$$

Now, bringing all the components together, we can observe how the input passes through the router and is directed to the selected experts. This process illustrates the complete flow within an MoE layer:



2.6—Sparse Routing and Top-k Selection

For large M, evaluating all experts is computationally prohibitive. Practical MoE variants use **sparse routing**, activating only the top-k experts per input. Let $\mathrm{S}(x) \subset 1, \ldots, M$ denote the selected experts, then

$$\hat{y}(x) = \sum_{i \in \mathrm{S}(x)} \tilde{g}_i(x) f_i(x),$$

where $\widetilde{g}_i$ are normalized gating weights.

Since top-k introduces non-differentiability, common strategies include:

1. **Sparse-softmax:** compute dense softmax logits, mask unselected experts, and propagate gradients.
2. **Noisy top-k:** add noise to logits before selection, enabling exploration and stable training.

2.7—Load Balancing

Without regularization, the router may collapse to a small subset of experts. To encourage balanced utilization, an auxiliary loss is added.

The **average gating probability** for expert j in a minibatch of size B is

$$\bar{g}_j = \frac{1}{B} \sum_{n=1}^{B} g_j(x^{(n)}; \phi).$$

A simple balancing regularizer is

$$\mathrm{L}_{\text{balance}} = \lambda \sum_{j=1}^{M} \left(\bar{g}_j - \frac{1}{M}\right)^2.$$

Thus, the total objective is

$$\mathrm{L}_{\text{total}} = \mathrm{L}_{\text{task}} + \mathrm{L}_{\text{balance}}.$$

More advanced methods penalize variance in both **importance** (sum of gating weights) and **load** (probability of selection), ensuring efficient capacity utilization.

3-EdgeAI Implementation

With this example you can implement the LLM in Raspberry Pi.