MoE 技术报告

(v1.0)

作者：纽约的自行车

起始时间：2025 年 01 月 16 号

目录

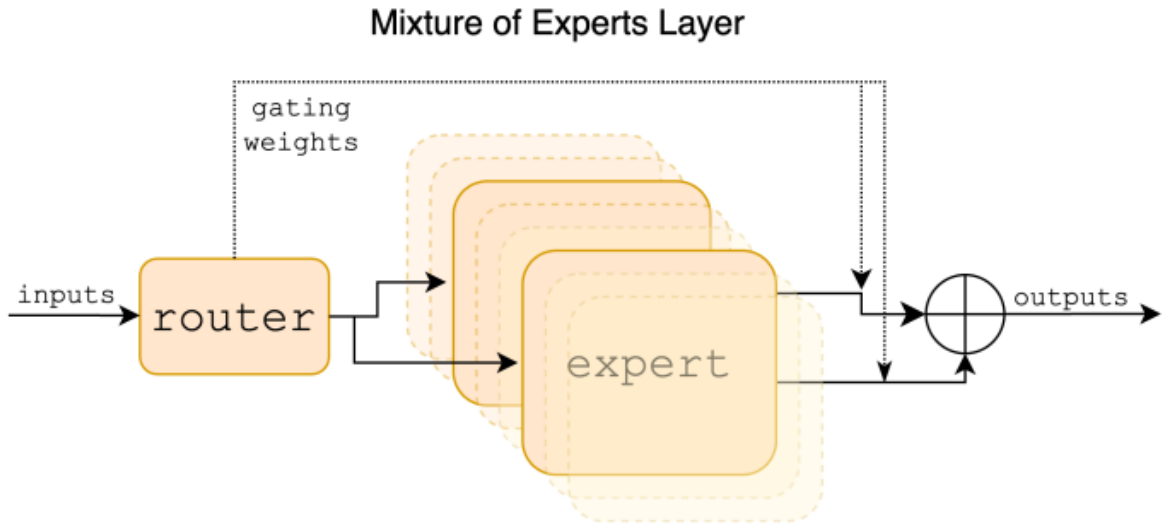
1 前言

MoE 的目的是模型稀疏化，每个输入 token 只激活一小部分模型参数。稠密模型具有 scaling law，这是通向 AGI 的关键，但是稠密模型每次推理都需要激活所有参数，无论是对于显存还是能量来说，推理成本都太大了。人脑是一个庞大的神经网络，但每次思考时只激活极少数的神经元，所以人脑是一个及其稀疏的神经网络，这样对人的消耗小且反应快。大模型也应该是一个稀疏网络，模型稀疏化是必经之路，但如何稀疏化是一个问题，目前普遍将 FFN 层稀疏化，称之为 MoE，后续对 attention 层稀疏化也是必然的。

2 MoE 架构

2.1 Mixtral MoE

最早将 MoE 引入到大模型的论文，采用常规的 token 选择专家方式，每个 token 选择 top-2 个专家，共有 8 个专家。



FFN 层改成 8 个专家，每次激活 2 个专家，每次推理激活 13B 参数，总参数 47B。使用 32k 长度数据训练。模型结构：

2.2 DeepSeek

DeepSeek 认为专家的参数量可以很小，专家数量可以很多，因此采用了细粒度专家方法。在保持参数数量不变的情况下，我们通过分割 FFN 中间隐藏维度将专家分割成更细粒度。相应地，在保持恒定的计算成本的情况下，我们还激活了更细粒度的专家，以实现激活专家的更灵活和适应性更强的组合。细粒度的专家分割允许将不同的知识更精细地分解，并更精确地学习到不同的专家中，每个专家都将保持更高的专业化水平。在专家数量有限的情况下，分配给特定专家的 token 更有可能涵盖不同类型的知识。因此，指定的专家将打算在其参数中学习截然不同的知识类型，而这些知识很难同时被利用。然而，如果每个 token 都可以路由到更多的专家，那么

Parameter	Value
dim	4096
n_layers	32
head_dim	128
hidden_dim	14336
n_heads	32
n_kv_heads	8
context_len	32768
vocab_size	32000
num_experts	8
top_k_experts	2

Table 1: Model architecture.

不同的知识将获得分别在不同专家中分解和学习的潜力。在这种情况下，每个专家仍然可以保持高水平的专家专业化，有助于在专家之间更集中地分配知识。我们通过将 FFN 中间隐藏维度减小到其原始大小的 1/m，将每个专家 FFN 分割成更小的专家。由于每个专家都变小了，作为回应，我们还将激活的专家数量增加到 m 倍，以保持相同的计算成本。

我们将某些专家隔离出来作为始终活动的共享专家，以捕捉和巩固不同情境下的共同知识。通过将公共知识压缩到这些共享专家中，其他路由专家之间的冗余将得到减轻。这样可以提高参数效率，确保每个路由专家都能专注于独特的方面，从而保持专业化。对于传统的路由策略，分配给不同专家的 token 可能需要一些公共知识或信息。结果，多个专家可以在获取其各自参数中的共享知识时收敛，从而导致专家参数的冗余。然而，如果有共享的专家致力于在不同的背景下获取和整合共同知识，那么其他路由专家之间的参数冗余将得到缓解。这种冗余度的减轻将有助于建立一个更具参数效率的模型，拥有更多专业专家。

此外，我们还调查了共享专家和路由专家的最佳比例。基于 64 名专家的最细粒度，并保持总专家和激活专家的数量不变，我们试图将 1、2 和 4 名专家隔离为共享专家。我们发现，共享专家和路由专家的不同比例对性能没有显著影响，1、2 和 4 个共享专家的桩损失分别为 1.808、1.806 和 1.811。考虑到 1:3 的比例会产生稍好的 loss 损失，当扩大 DeepSeekMoE 时，我们将共享专家和激活的路由专家之间的比例保持为 1:3。

2.3 Arctic-MoE

Arctic 使用了一种独特的 Dense MoE 混合架构。它将 10B dense 模型与残差 128×3.66B MoE MLP 相结合，使用前 2 个门控选择总共 480B 和 17B 个激活参数。

在设计 Arctic 时，我们注意到模型质量的提高主要取决于专家的数量和 MoE 模型中的参数总数，以及这些专家可以组合在一起的方式的数量。基于这一见解，Arctic 被设计为在 128 位细

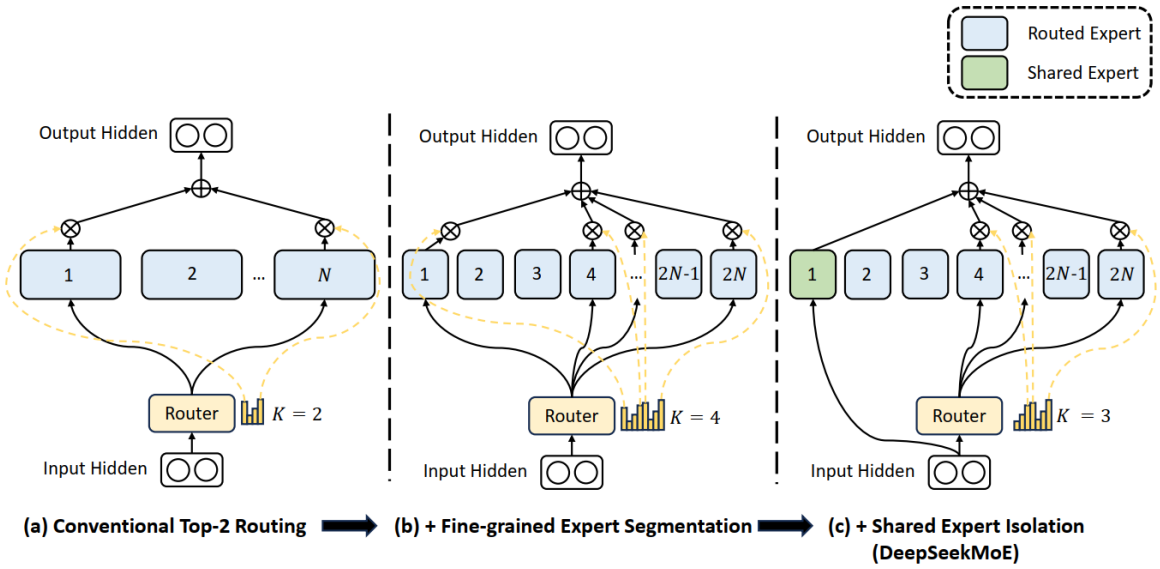
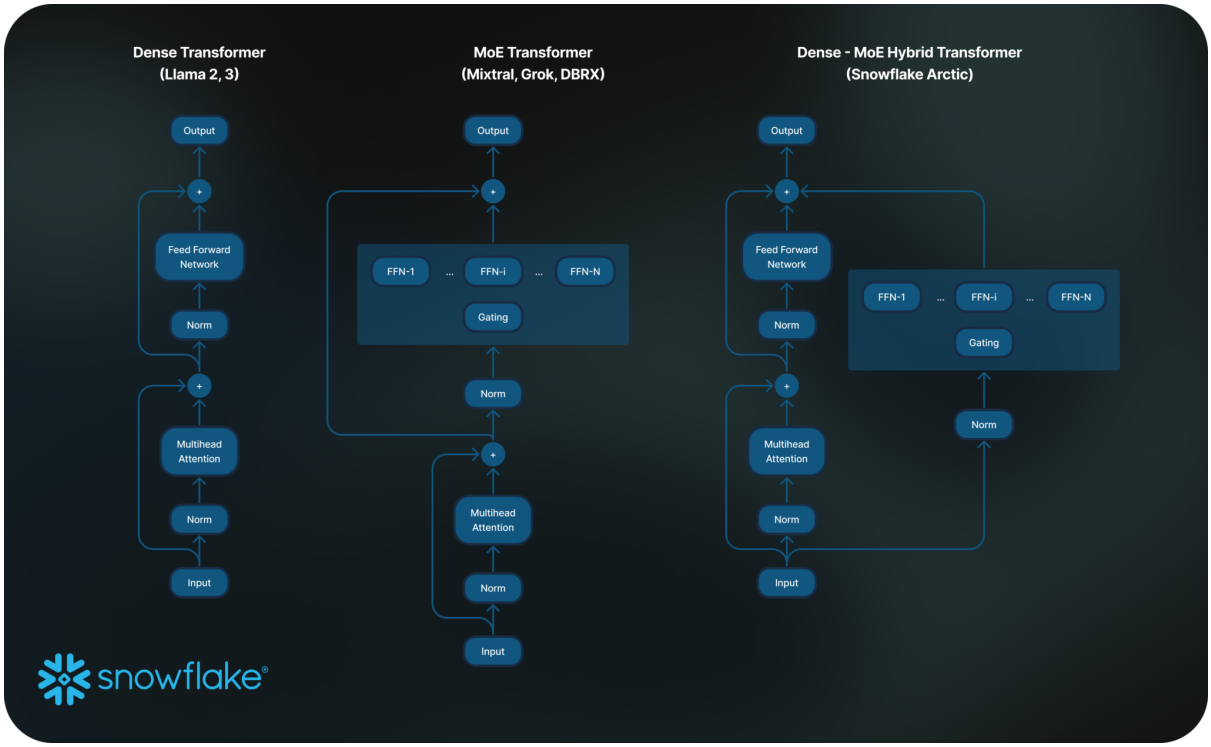


Figure 2 | Illustration of DeepSeekMoE. Subfigure (a) showcases an MoE layer with the conventional top-2 routing strategy. Subfigure (b) illustrates the fine-grained expert segmentation strategy. Subsequently, subfigure (c) demonstrates the integration of the shared expert isolation strategy, constituting the complete DeepSeekMoE architecture. It is noteworthy that across these three architectures, the number of expert parameters and computational costs remain constant.



粒度专家中分布 480B 参数，并使用前 2 个门控选择 17B 激活参数。我们的第二个见解是，在 Arctic 架构中将 Dense 与残余 MoE 组件相结合，使我们的训练系统能够通过通信计算重叠实现良好的训练效率，从而隐藏了大部分通信开销。

2.4 DeepSpeed-MoE

动机 1 及实验：a）我们将 MoE 层放在模型的前半层中，并使后半层与 Dense 模型相同（称为 First-Half-MoE），以及 b）我们将 MoE 层切换到后半层，并在前半层使用 Dense（称为 Second-Half-MoE）。结果如图 2（左）所示。可以看出，Second-Half-MoE 的表现明显优于同行。这证实了并非所有 MoE 层都学习相同级别的表示。深层受益更多大量专家。

动机 2：为了提高 MoE 模型的泛化性能，有两种常见的方法：（1）增加专家数量，同时保持专家容量（即每个令牌所经过的专家数量）不变；（2）在保持相同数量的专家的同时，以略多的计算（33%）为代价使专家能力翻倍。然而，对于（1），由于专家人数较多，需要增加训练资源的内存需求；对于（2），更高的容量也会使通信量翻倍，这会显著减慢训练和推理的速度。有没有一种平衡方法?

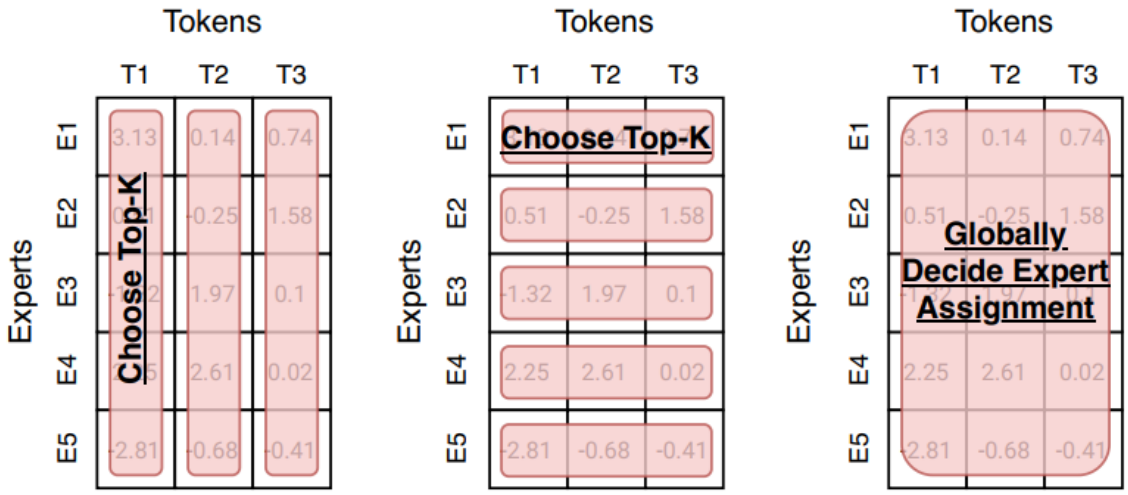
实验：为什么更大的专家容量有助于准确性的一个直觉是，这些额外的专家可以帮助纠正第一个专家的“表示”。然而，第一个专家是否每次都需要更改？或者我们可以修复第一个专家，只将不同的额外专家分配给不同的令牌吗？为了研究这一未知性质，我们以两种方式进行了比较：（1）将容量增加一倍（称为 Top2 MoE），以及（2）固定一名专家，并在不同专家之间改变第二名专家（称为剩余 MoE）。主要的直觉是将来自 MoE 模块的专家视为密集 MLP 模块的纠错项。结论：我们发现这两种方法（即 Top2-MoE 和残差 MoE）的泛化性能是不相上下的。然而，由于通信量的减少，我们的新设计残差 MoE 的训练速度比 Top2 MoE 快 10

在此基础上，我们提出了一种新颖的 MoE 体系结构。正如动机 1 所表明的那样，在后几层

利用 MoE 带来了更多的好处，与前几层相比，我们的新架构在最后几层使用了更多的专家。这给出了金字塔 MoE 的设计，同时，考虑到动机 2，我们提出了残差 MoE 架构，其中每个令牌分别通过一个固定的 MLP 模块和一个选定的专家。

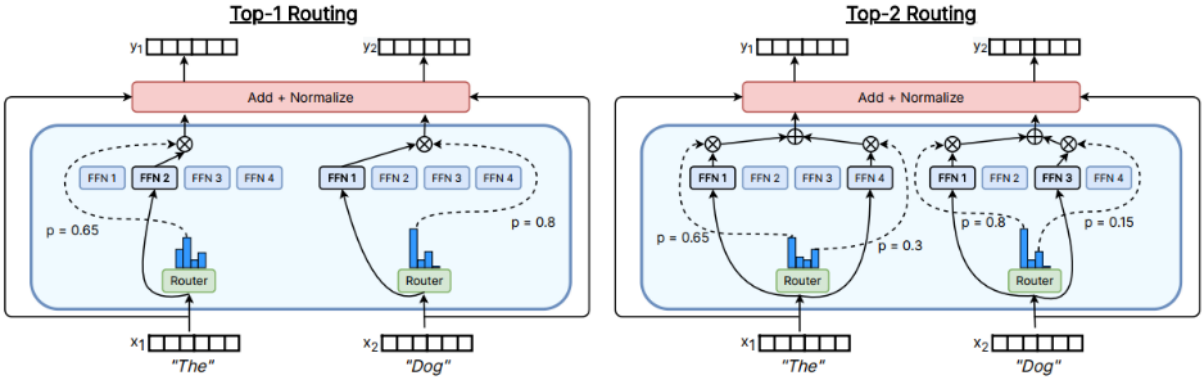
3 路由算法

路由方式有三种：1、由 token 自己决定选择哪个专家；2、由专家去选择 token；3）全局决定哪些 token 应该分配到哪些专家，不是由 token 做决定也不是专家做决定。



3.1 token 决定路由

下图展示了 token 决定路由到哪个专家的过程，左图是每个 token 只路由到一个专家，右图是每个 token 可路由到 2 个专家。这种方式下每个 token 都会被路由到固定数量的专家，不会有 token 被遗漏，但是每个专家接收的 token 数量不一致，会导致专家复杂不均衡。



上图是从 transformer 结构层面展示路由过程，下图是从具体计算过程展示路由过程。Router Weights 是一个矩阵，由 5 个专家组成，这里可以看出所谓的路由网络就是一个矩阵，而专家就是一个小小的 FFN 层网络。图中有三个 token，最下面展示了 token 和路由矩阵相乘后，token 路由到专家的概率。T1 token 被路由到专家 1，T2 token 被路由到专家 4，T3 token 被路由到专家 2。

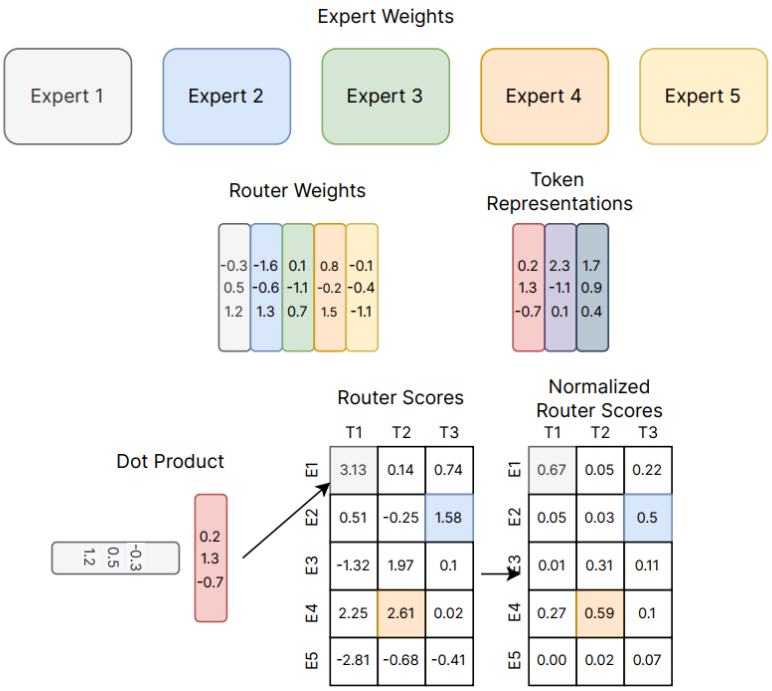
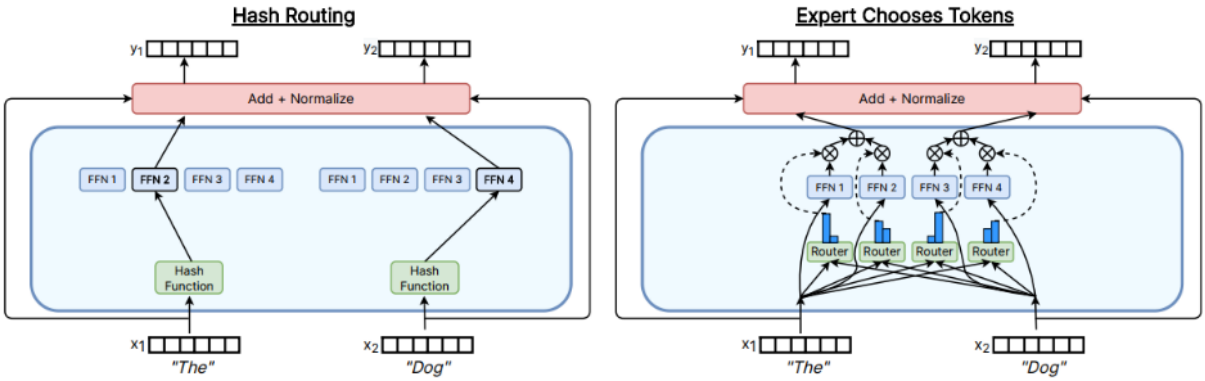


Figure 2: **Schematic of top-k routing**. We visualize an example of the top-k token routing scheme over five experts and three input tokens. Each expert and token is color-coded and the router weights (W_r) have a representation for each expert (color matched). To determine the routing, the router weight performs a dot product with each token embedding (x) to produce the router scores ($h(x)$). These scores are then normalized to sum to one ($p(x)$).

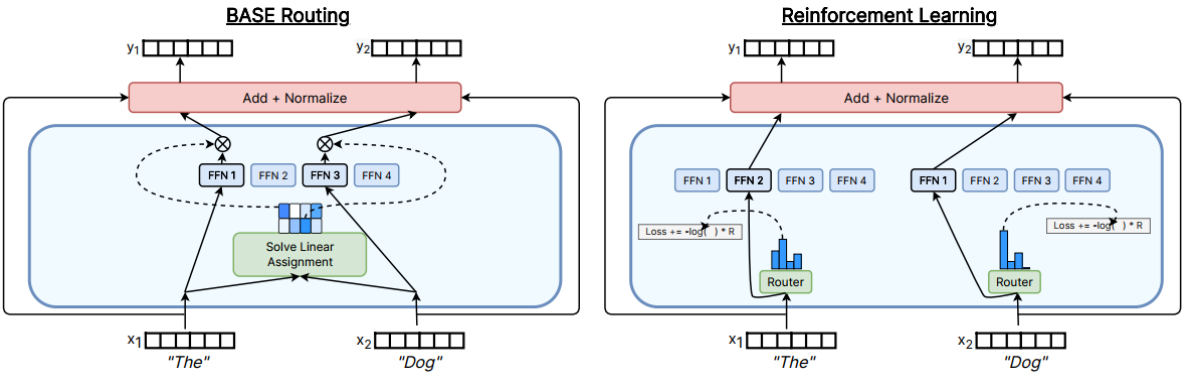
3.2 专家决定路由

下图展示了专家决定 token 选取的过程，每个专家和每个 token 计算一个匹配概率，每个专家选取 top-k 个和自己最匹配的 token。这种方式下专家接收的 token 数量是固定的，不存在专家复杂不均衡问题，但是存在 token 丢失的现象，也就是 token 没有被任何一个专家接收。



3.3 全局决定路由

设计一个算法来决定路由，这种方式称为全局路由。常用的算法由 BASE、Hash 和强化学习，但这种方式在目前的 MoE 模型中使用少。



4 Dense 改造成 MoE

4.1 Qwen-MoE

DeepSeek-MoE 和 DBRX 已经证明了 finegrained experts 的有效性。从 FFN 层过渡到 MoE 层时，我们一般只是简单地复制多次 FFN 来实现多个 expert。而 finegrained experts 的目标是在不增加参数数量的前提下生成更多 expert。为了实现这一点，我们将单个 FFN 分割成几个部分，每个部分作为一个独立的 expert。我们设计了具有总共 64 个 expert 的的 MoE，对比其他配置，我们认为这个实现能达到效果和效率的最优。

模型初始化阶段至关重要。初步实验表明，从零开始训练 MoE 模型可能效率低下，且难以提升至预期的最优性能水平。因此，我们首先利用已有的 Qwen-1.8B，将其改造为 Qwen1.5-MoE-A2.7B。此外，在初始化阶段引入随机性可以显著加快收敛速度，并在整个预训练过程中带来更好的整体性能表现。

目前，一个明显的趋势是在 MoE 中实现共享 expert 与 routing expert。从更宏观的角度看，这是一种广义的 routing 方法，因为在没有共享 expert 的情况下，实际上就退化为传统的 MoE 路由设置。对于 Qwen1.5-MoE-A2.7B 模型，我们在其中整合了 4 个总是被激活的共享 expert 和每次只激活其中 4 个的 60 个 routing expert。这种方式非常灵活，同时在我们实验中效率最佳。

相比 Qwen1.5-7B，Qwen1.5-MoE-A2.7B 的训练成本降低了 75%，推理速度则提升至 1.74 倍。

初始化方法猜测：1. 先按照 Qwen1.5-1.8B 的 intermediate_size 5504 进行分割，分割成 4 个小的 expert，每个 expert 是 1376 维。然后再加入随机性，加上随机初始化的 32 维，变成 1408 维。其余非 moe 的参数，就直接继承 Qwen1.5-1.8B；2. layer 0 和 layer >= 19 层应该是随机初始化的，而其他层权重是从 Dense 复制的；

对 Dense FFN 随机打乱顺序是合理的。FFN 相邻维度之间并没有相关性，token 是和 FFN 的单个维度相关的，所以从 dense 初始化的最佳专家维度是 1。打乱顺序后，改造成 MoE，每个专家都是从原 FFN 不连续的维度组成。专家粒度越细 top-k 越大，效果越好。

如果 Dense FFN 是稀疏的，即每个 token 实际上只需要使用少量 FFN 即可，这种 Dense 是最适合用来改造成 MoE 的。

将一个大的 Dense 模型改造成 MoE。原 FFN 层不用复制，先随机打乱 FFN 维度顺序，FFN 的 up 和 down 矩阵需要按照同样的顺序打乱。然后切分成极细粒度专家。

4.2 Skywork MoE

这篇论文详细阐述了他们是如何将一个 Dense 模型改造成 MoE 模型。Skywork MoE，一个拥有 1460 亿个参数和 16 位专家的高性能 MoE 大型语言模型。该模型利用之前开发的 Skywork-13B 模型的基础架构，利用其 checkpoint 作为初始设置。Skywork MoE 融合了两种新的训练技术：gating logit normalization 和 adaptive auxiliary loss coefficients，体现了 MoE 研究的前沿。前者旨在增强专家之间的多样性，而后者有助于在模型的不同层对辅助损失系数进行量身定制的调整。Skywork MoE 经历了几个阶段的训练，每个阶段都有独特的学习率和数据组成。用于训练 SkyworkMoE 的数据由 SkyPile 语料库的一个子集组成，包含大量的合成数据。总体而言，训练数据的总体分布在英语、中文和代码数据之间的比例约为 7:2:1。

（1）Upcycling vs From Scratch

从 Dense 模型升级改造成 MoE，还是从零预训练 MoE，这是个问题。先说结论：预算充足就从零预训练，否则从 Dense 改造性价比更高。

为了做实验对比，作者在 300B tokens 上训练了一个 0.3B 规模的 Dense 模型，从 checkpoint 中选择了训练 100B 和 300B tokens 的模型。实验用的 MoE 是 8 个专家，三种初始化方式：随机初始化、checkpoint-100B、checkpoint-300B，对比了不同学习率，对比了继续训练 100B 和 300B 的效果，实验结果如下图：

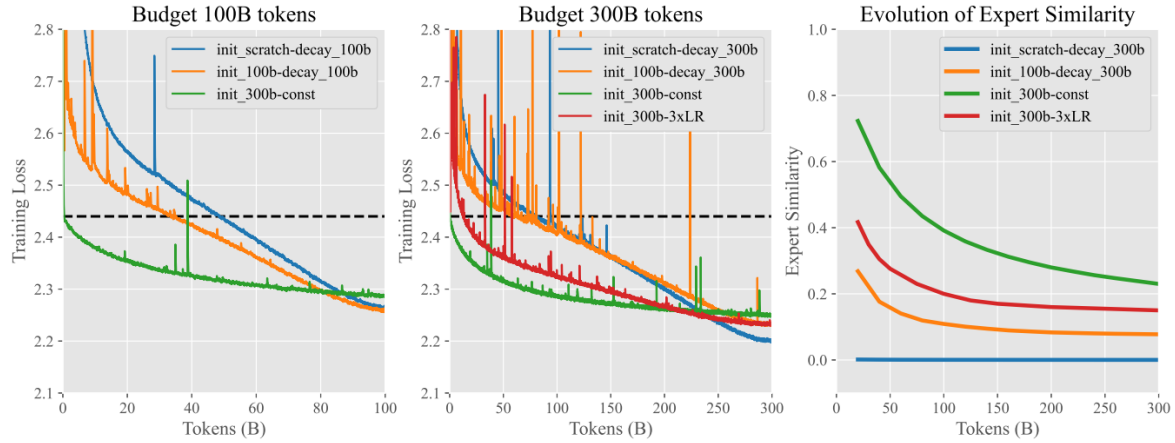


Figure 1: Training dynamics under different conditions and budgets. Left: Loss curves for MoE training initialized by upcycling and from scratch with 100B token budget. Middle: Similar comparison for a 300B token budget. Right: Evolution of average expert similarity during MoE training with a 300B token budget. The dashed line marks the final loss of a 0.3B dense model at the end of 300B tokens.

图 1 最左侧图，继续训练 100B tokens，init_100b-decay_100b 和 init_scratch-decay_100b 效果最终相同，说明在训练资源有限情况下，从 Dense 改造的方法是可行的。至于 init_300b-const 效果差，作者归因于其过小学习率 3e-4。图 1 中间图，继续预训练 300B tokens，蓝色线 loss 最低说明从零预训练效果最好。其他曲线中，学习率最小的效果最差。说明采用 Dense 改造的方式需要注意学习率。图 1 最右侧图，专家相似度越大效果越差。意味着采用 Dense 改造的方式，要注意训练过程中专家之间是否开始具备差异化，只有开始差异化后效果才会变好。

（2）Expert Specialization Training for Upcycling

将 Dense 改造成 MoE 架构，常用方法是将原 FFN 层复制 n 份，每份当做一个专家。这种

做法的一个潜在的缺点是专家初始化时都是相同的，没有差异性。前文提到专家的差异性对效果至关重要，因此作者实验验证这种方法是否有效。作者在 1T tokens 上训练了一个 1.3B 模型 Mbase，然后将 Mbase 分别单独在 100B 的中文、英文、代码数据上训练，此时仅更新 FFN 层参数，得到三个模型 Mcn、Men、Mcode。Baseline：将 Mbase 的 FFN 层复制 8 份，作为 8 个专家；Multi. Init.：复制三份 Mcn 中的 FFN，复制三份 Men 中的 FFN，复制两份 Mcode 中的 FFN，将这 8 份 FFN 组成 MoE 层，其余参数来自 Mbase 模型。

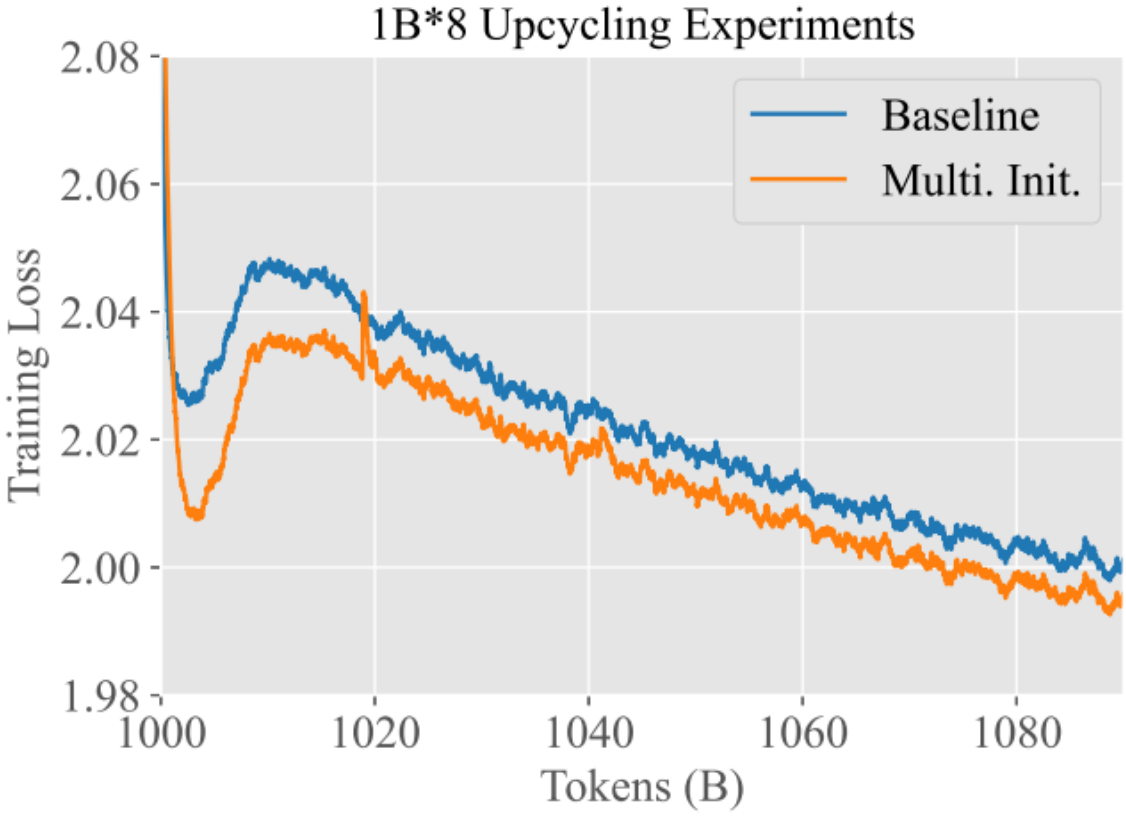


Figure 8: Comparison of training loss for MoE models: conventional upcycling (baseline) versus specialization training (Multi. Init.). Both models underwent training over 100 billion tokens.

从上图来看，训练 90B 数据后，两种初始化方式的差异可忽略，因此作者认为专家初始化时的差异性并不重要。

4.3 Sparse Upcycling

在本文中，我们探讨了模型升级：用相对较小的额外计算预算升级现有模型。特别是，我们专注于将 Dense 模型升级为更大、稀疏激活的混合专家（MoE）。稀疏的升级回收在两种情况下可能特别有价值：（i）一个人可以使用预训练的 Transformer（有许多公开可用的），并希望用适度或有限的计算预算对其进行改进。（ii）有人计划训练一个大型模型，但不知道 Dense 模型还

是 MoE 模型更有效：可以通过首先训练 Dense 模型，然后在 Dense 模型饱和后将其升级为 MoE 模型来同时训练两者。

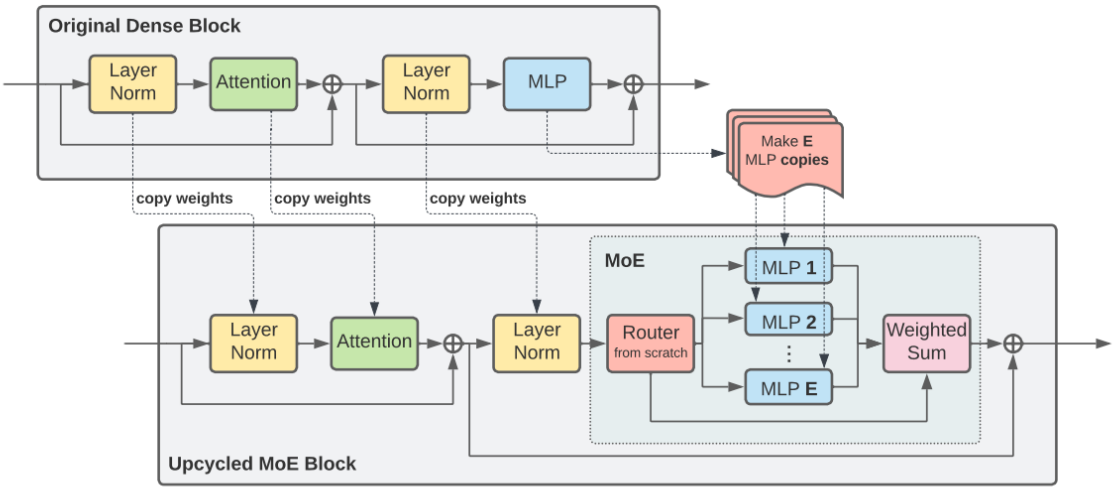


Figure 1: The upcycling initialization process. All parameters, and optionally their optimizer state, are copied from the original checkpoint, except those corresponding to the MoE router, which does not exist in the original architecture. In particular, the experts in the new MoE layer are identical copies of the original MLP layer that is replaced.

每个 MoE 层都包含固定数量的专家。每个专家都被初始化为原始 MLP 的副本。此外，我们添加了一个权重随机初始化的路由器。通过增加上循环层的数量、专家数量或专家容量来增加模型容量通常会导致更高质量的模型，但由于层的更剧烈的重新配置，也会增加计算成本和/或导致更大的初始质量下降。

另一个关键决定是要稀疏化多少层。更多的层导致更高的模型容量，同时它引入了显著的额外时间开销。我们发现稀疏化第一层往往是有问题的。Dense 总共 12 层，改造后面一半层为 MoE 效果最好。

5 专家负载均衡

专家负载均衡问题在论文：Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity 有详细描述。

上图是一个 token 决定路由的 MoE 模型，在这种路由方式下，会出现每个专家接的 token 数量不一致情况，而且很常见。训练中可以观察到，路由网络倾向于收敛到这样一种状态，即它总是为同样少数的专家产生大的权重。这种不平衡是自我强化的，因为受青睐的专家被更快地训练，从而被路由网络更多地选择。为了使各个专家都能被同等对待，可以定义专家重要性损失，例如将一个 batch 中某个专家的路由概率之和称为专家重要性。这个损失的目的就是让各个专家都具有相同的重要性。然而，一些专家可能以高概率接收了小部分 token，另一些专家以小概率接收了大量 token，这样总的重要性差不多，但是每个专家训练的 token 数量仍然差距大。

在 Switch Transformer 论文中，作者提出一种新的专家负载均衡损失。假设由 N 个专家，一个 batch B 中由 T 个 token，负载损失 loss 如下：

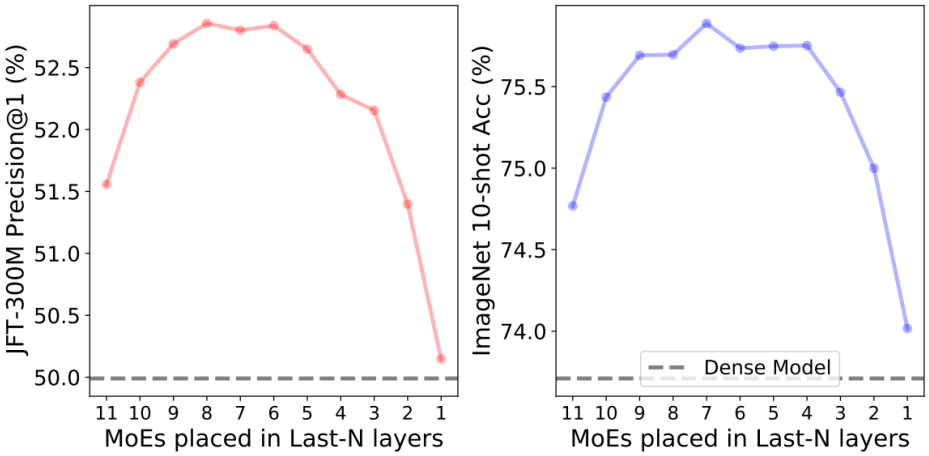


Figure 12: Performance as a function of the number of MoE layers for upcycled B/16 models ($C = 1$) trained for 7 additional epochs on JFT, starting from a dense checkpoint originally trained for 14 epochs. MoE layers are consecutively placed starting from the last block. We train models ranging from 1 MoE layer (Last-1) to 11 MoE layers (Last-11) – i.e. all but the very first. The dashed horizontal lines show the performance of the dense model when trained for an additional 7 epochs.

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

fi 表示专家 i 分配的 token 数量占全部 token 数量的比例。Pi 表示分配给专家 i 的 token 的概率平均值。这个损失函数很奇怪，理想值是 top-k 中的 k 而不是 0。

举例说明：

假设有 4 个专家，激活 top-2，batch 中共有 4 个 token。

—

分布情况（1）：pij 表示 token i 路由到专家 j 的概率

p11=0.8, p12=0.2, p13=0, p14=0

p21=0.7, p22=0.3, p23=0, p24=0

p31=0.6, p32=0.4, p33=0, p34=0

p41=0.4, p42=0.6, p43=0, p44=0

根据上述概率，4 个 token 均被路由到专家 1 和 2，计算负载 loss：

f1=f2=1, f3=f4=0

P1=(0.8+0.7+0.6+0.4)/4=0.625

P2=(0.2+0.3+0.4+0.6)/4=0.375

P3=P4=0

aux_loss = 4*(1*0.625 + 1*0.375) = 4

—

分布情况（2）：pij 表示 token i 路由到专家 j 的概率

p11=0.7, p12=0.2, p13=0.1, p14=0

p21=0.5, p22=0.3, p23=0.2, p24=0

p31=0.4, p32=0.4, p33=0.2, p34=0

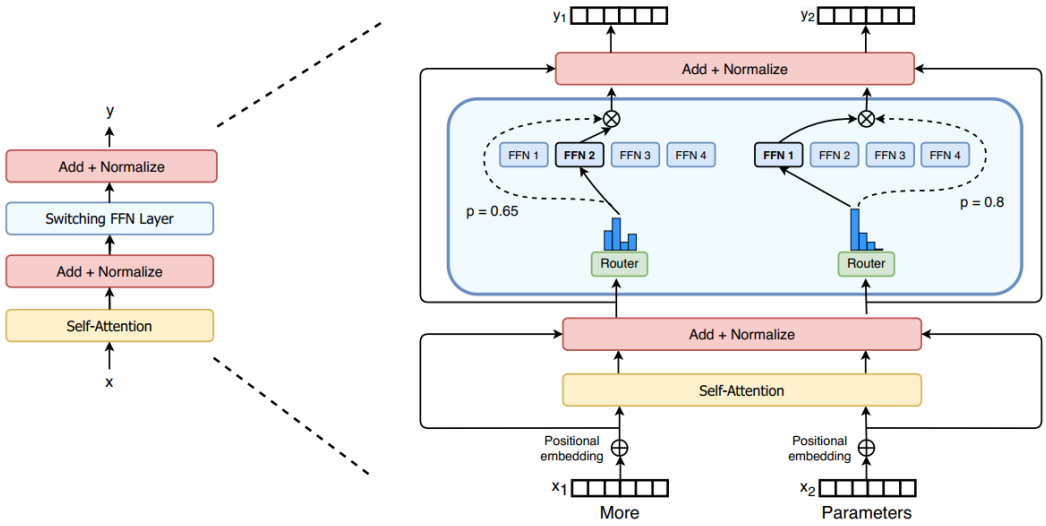


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

p41=0.4, p42=0.4, p43=0.2, p44=0

根据上述概率，4 个 token 均被路由到专家 1 和 2，计算负载 loss：

f1=f2=1, f3=f4=0

P1=(0.7+0.5+0.4+0.4)/4=0.5

P2=(0.2+0.3+0.4+0.4)/4=0.325

P3=(0.1+0.2+0.2+0.2)/4=0.175

P4=0

aux_loss = 4*(1*0.5 + 1*0.325) = 3.3

—

分布情况（3）：pij 表示 token i 路由到专家 j 的概率

p11=0.7, p12=0.2, p13=0.1, p14=0

p21=0.5, p22=0.3, p23=0.2, p24=0

p31=0, p32=0.2, p33=0.4, p34=0.4

p41=0, p42=0.2, p43=0.5, p44=0.3

根据上述概率，4 个 token 均被路由到专家 1 和 2，计算负载 loss：

f1=f2=1=f3=f4=0.5

P1=(0.7+0.5)/4=0.3

P2=(0.2+0.3+0.2+0.2)/4=0.225

P3=(0.1+0.2+0.4+0.5)/4=0.3

P4=(0.4+0.3)/4=0.175

aux_loss = 4*(0.5*0.3 + 0.5*0.225 + 0.5*0.3 + 0.5*0.175) = 2

—

分布情况（4）：pij 表示 token i 路由到专家 j 的概率

p11=0.5, p12=0.5, p13=0, p14=0

p21=0.5, p22=0.5, p23=0, p24=0

p31=0, p32=0, p33=0.5, p34=0.5

p41=0, p42=0, p43=0.5, p44=0.5

根据上述概率，4 个 token 均被路由到专家 1 和 2，计算负载 loss：

f1=f2=1=f3=f4=0.5

P1=(0.5+0.5)/4=0.25

P2=(0.5+0.5)/4=0.25

P3=(0.5+0.5)/4=0.25

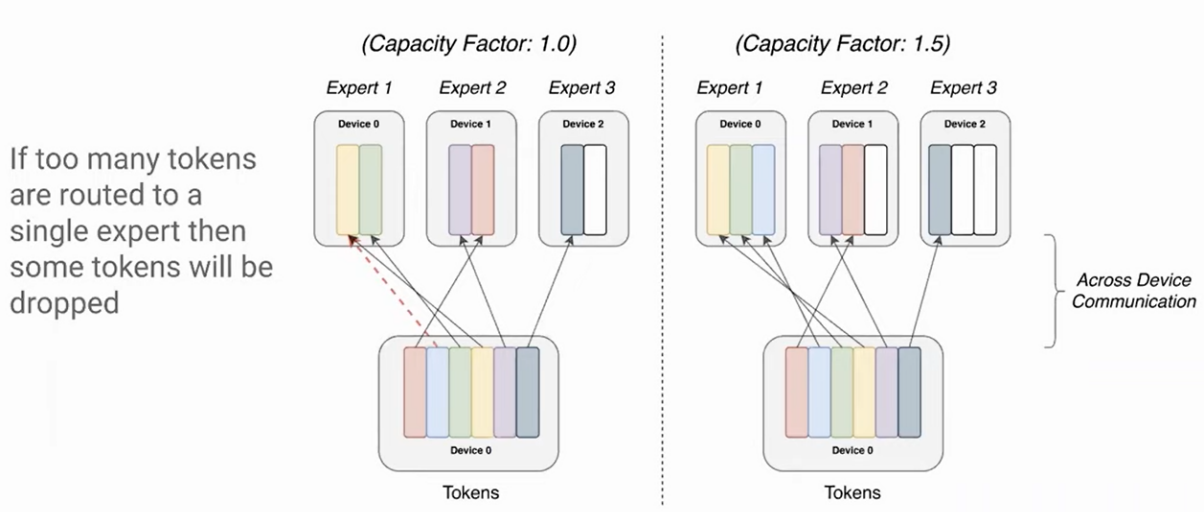
P4=(0.5+0.5)/4=0.25

aux_loss = 4*(0.5*0.25 + 0.5*0.25 + 0.5*0.25 + 0.5*0.25) = 2

从分布 3、4 分析，只要专家有接收到 token，那么计算 loss 时该专家对所有 token 的路由概率都会被计算进来，导致负载 loss 总是在 k 附近。从分布 1、2 来看，因为有 2 个专家没有接收到 token，那么这两个专家对所有 token 的路由概率都不会计入到 loss 中，这部分数值被浪费了，导致 loss 偏离 k。

这个损失是好还是不好？从 MoE 实际负载结果分析，训练时负载损失很快稳定在 k，但是各层各个专家的负载并没有均衡，所以这个损失并不好。

6 Token 丢失解释

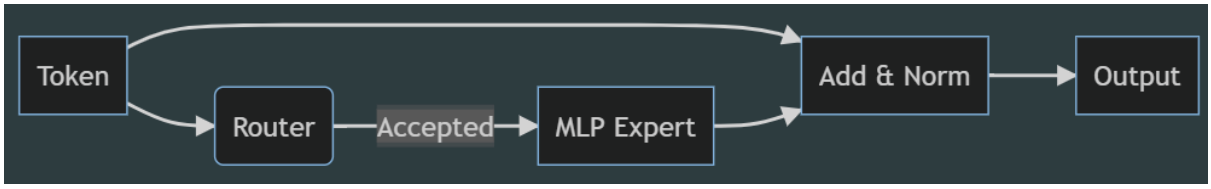


专家并行推理或者训练时，需要设置专家容量（EC，expert capacity）系数，

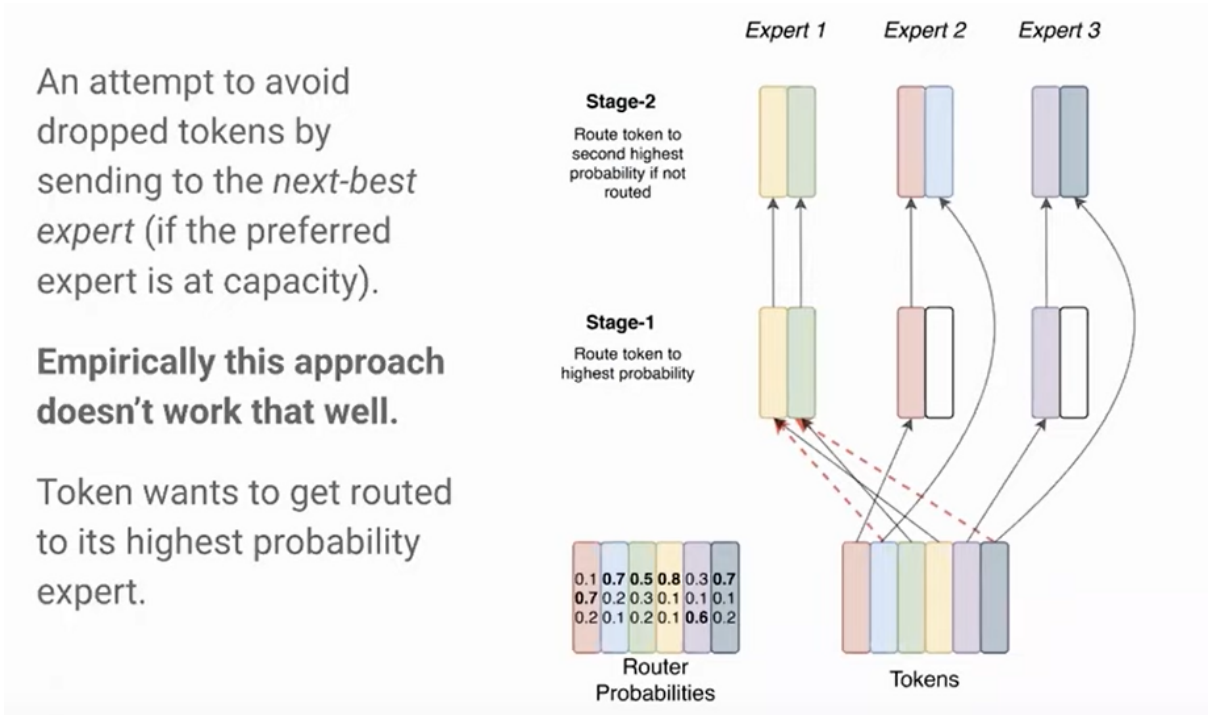
$$EC = round\frac{CF * k * BS_{tokens}}{E}$$

。CF，capacity factor 表示容量因子，是一个超参。k 表示 top-k，BS 表示 tokens per batch 的数量，E 表示专家数量。当 CF<1 时，总会丢弃一些 token。当 CF>1 时，会增加闲置容量，并减少丢弃 token 的频率。在实践中，某些 token 将比其他 token 更依赖于一些专家，并且由于分配无限高的 CF 是不可行的，一些 token 将不得不放弃。当路由到专家的 token 数量超过其 EC 时，MoE 层将简单地处理一些 token。

如上图所示，当 EC=1 时，top-2 路由选择导致每次每个专家最多计算两个 token，当 EC=1.5 时，top-2 路由选择就会每次给每个专家最多计算三个 token。如果这对你来说是不直观的（token 去哪里了？），请记住，transformer 在使用前一个输入的每个 FFN/Attn 层之后都会进行 add 和 norm 运算，所以当前层丢弃 token 就是一个空操作：



我们不是让 token 不被处理，而是至少可以尝试通过将它们发送给 router 确定的第二、第三、第 n 个最有可能的专家来处理它们吗？经验表明这种方法并不起作用，实际上 token 最喜欢被路由到概率最大的专家。



MoE 实际上不需要处理所有 token 即可工作；具有相当低的 CF 和大量的 token 丢弃是可以的。

Model	Capacity Factor	Quality after 100k steps (Neg. Log Perp.)	Time to Quality Threshold (hours)	Speed (examples/sec)
T5-Base	—	-1.731	Not achieved†	1600
T5-Large	—	-1.550	131.1	470
MoE-Base	2.0	-1.547	68.7	840
Switch-Base	2.0	-1.554	72.8	860
MoE-Base	1.25	-1.559	80.7	790
Switch-Base	1.25	-1.553	65.0	910
MoE-Base	1.0	-1.572	80.1	860
Switch-Base	1.0	-1.561	**62.8**	1000
Switch-Base (expand)	1.0	**-1.534**	67.6	780