第一章　模型结构介绍

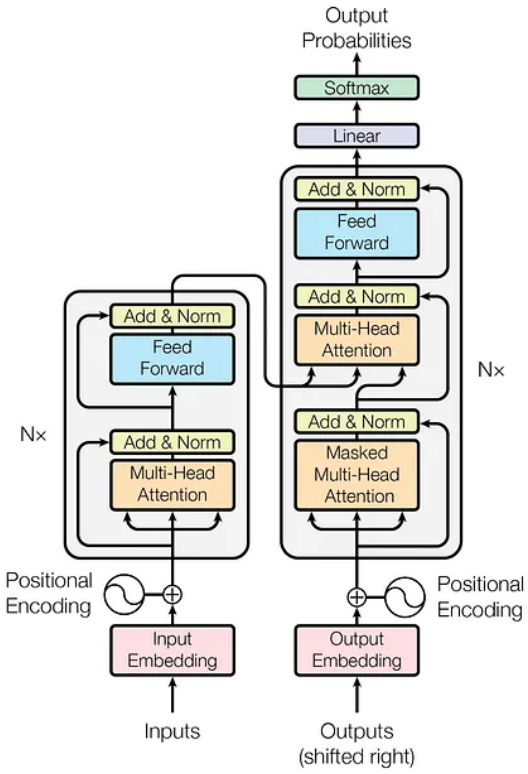


图 2.1 Transformer 模型结构

Transformer 模型由编码器和解码器组成，每个层包含两个核心组件：**注意力机制（Attention）和前馈神经网络（FFN）**。注意力机制负责捕捉序列中不同位置之间的依赖关系，而 FFN 则对每个位置的表示进行非线性变换，增强模型的表达能力。

第1条　Attention（注意力机制）

注意力机制是 Transformer 的核心组件，通过计算序列中不同元素之间的相关性来捕捉全局依赖关系。

第2条　基本原理

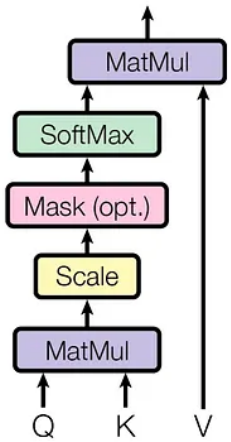


图 2.2 Attention 计算图

注意力函数通过三个输入矩阵进行计算：**查询（Query, Q）**、**键（Key, K）和值（Value, V）**。查询表示当前需要关注的位置，键表示序列中所有位置的表示，值则是需要提取的信息表示。

注意力计算过程如下：

1) **注意力分数计算：**通过查询矩阵Q与键矩阵K的转置进行矩阵点积，得到注意力分数矩阵。矩阵中元素a_{ij}表示位置i#对位置j#的关注程度。

2) **缩放与归一化：**将注意力分数除以$\sqrt{d_k}$（其中d_k为键的维度），以防止梯度过大，然后应用Softmax函数进行归一化，使得每行的权重和为1。

3) **加权求和：**将归一化后的注意力权重矩阵与值矩阵V相乘，得到最终的注意力输出。

数学表达式为：

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

第3条　注意力机制变体

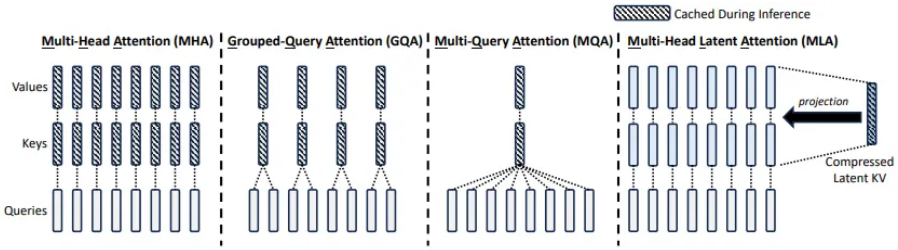
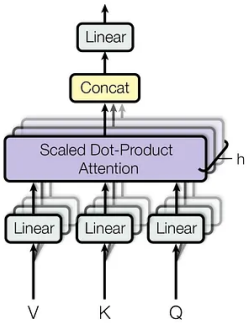


图 2.3 Attentions

1) **MHA（Multi-Head Attention，多头注意力）**传统的多头注意力机制将输入投影到h#个不同的子空间，并行计算h#个独立的注意力头，然后将结果拼接并再次投影。每个头使用独立的投影矩阵W_i^Q、W_i^K、W_i^V（$i = 1,2,\ldots,h$），使得模型能够从不同表示子空间捕捉信息。



在原始Transformer中，通常设置$h=8$，每个头的维度$d_k = d_v = d_{model}/h = 64$。多头注意力的总计算成本与单头全维注意力相近，但表达能力更强。

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

2) **MQA (Multi-Query Attention，多查询注意力)**多查询注意力机制使所有头共享相同的键和值投影矩阵，仅查询矩阵保持独立。这种设计显著减少了KV Cache的内存占用，在解码阶段可将推理速度提升约13.9倍（参考论文：Multi-Query Attention）。

3) **GQA (Grouped-Query Attention，分组查询注意力)**分组查询注意力是MQA和MHA之间的折中方案，将多个头分组，每组共享一个键值投影矩阵。例如，在T5-XXL（64个头）中，可将64个头分为8组，每组8个头共享键值投影。相比传统MHA，GQA在保持模型质量的同时，推理速度提升5-6倍（参考论文：Grouped-Query Attention）。

4) **MLA (Multi-head Latent Attention，多头潜在注意力)**MLA是DeepSeek-V3采用的注意力机制，通过将键值矩阵合并压缩为低秩向量来减少KV Cache的存储需求。压缩维度远小于MHA中的投影矩阵维度（可从7168维压缩至256维，节省约28倍），从而显著降低内存占用。（参考论文：Multi-head Latent Attention）

Keys和Values的低秩压缩与RoPE解耦：

$c_t^{KV} = W_D^{KV} h_t$\$

$[k_{t,1}^C; k_{t,2}^C; \cdots; k_{t,n_h}^C] = k_t^C = W_U^K c_t^{KV}$\$

$k_t^R = \text{RoPE}(W_K^R h_t)$\$

$k_{t,i} = [k_{t,i}^C; k_t^R]$\$

$[v_{t,1}^C; v_{t,2}^C; \cdots; v_{t,n_h}^C] = v_t^C = W_U^V c_t^{KV}$\$

其中$c_t^{KV} \in \mathbb{R}^{d_c}$是Key和Value的共享压缩低秩向量（$d_c \ll d_h \cdot n_h$），$W_D^{KV}$是下采样矩阵，$W_U^K$和$W_U^V$是上采样矩阵。通过这种设计，Key的内容信息$k_{t,i}^C$和Value内容信息$v_{t,i}^C$共享同一个低秩向量$c_t^{KV}$，而位置信息则通过独立的RoPE路线$k_t^R = \text{RoPE}(W_K^R h_t)$生成，确保位置信息不被压缩损失。最终第$i$#个头的Key为$k_{t,i} = [k_{t,i}^C; k_t^R]$（内容和位置的拼接）。

Queries的低秩压缩与RoPE解耦：

$c_t^Q = W_D^Q h_t$\$

$$[q_{t,1}^C; q_{t,2}^C; \cdots; q_{t,n_h}^C] = q_t^C = W_U^Q c_t^Q \$$

$$[q_{t,1}^R; q_{t,2}^R; \cdots; q_{t,n_h}^R] = q_t^R = \text{RoPE}(W_Q^R c_t^Q) \$$

$$q_{t,i} = [q_{t,i}^C; q_{t,i}^R] \$$

其中$c_t^Q \in \mathbb{R}^{d_c'}$是Query的压缩低秩向量（$d_c' \ll d_h \cdot n_h$），$W_D^Q$是下采样矩阵，$W_U^Q$是上采样矩阵。与Keys/Values类似，Query的内容信息$q_{t,i}^C$由压缩低秩向量解压得到，而每个头的位置信息$q_{t,i}^R$通过 RoPE$(W_Q^R c_t^Q)$独立生成，最终$q_{t,i} = [q_{t,i}^C; q_{t,i}^R]$。

注意力计算与输出：

$$o_{t,i} = \sum_{j=1}^{t} \text{Softmax}_j\left(\frac{q_{t,i}^T k_{j,i}}{d_h + d_h^R}\right) v_{j,i}^C \$$

$$u_t = W_O[o_{t,1}; o_{t,2}; \cdots; o_{t,n_h}] \$$

其中缩放因子$d_h + d_h^R$由内容维度d_h和RoPE维度d_h^R组成，$W_O \in \mathbb{R}^{d \times d_h \cdot n_h}$是输出投影矩阵。通过这种设计，只需缓存$c_t^{KV}$和$k_t^R$两个张量，相比标准MHA大幅减少KV Cache占用。

第4条　Embedding（词嵌入层）

Embedding 层是模型的入口，负责将输入 token ID 转换为高维向量表示。

参数：Embedding $\in \mathbb{R}^{V \times d}$，其中$V$#为词表大小，$d$#为隐层维度

计算：本质上是一个**查表(Gather)操作**，无浮点运算

作用：为后续 Transformer 层提供初始输入表示

对于 DeepSeek-V3：vocab_size = 129280，hidden_size = 7168，参数量约 0.9GB（FP8）。

第5条　LMHead（语言模型头）

LMHead 是模型的出口，将最后一层的隐状态投影回词表空间，用于生成 token 的预测概率。

参数：LMHead $\in \mathbb{R}^{d \times V}$，通常与 Embedding 权重共享以节省显存

计算：矩阵乘法，计算量为$2 \times d \times V$

作用：输出 logits，通过 softmax 获得 token 分布用于采样

对于 DeepSeek-V3：参数量约 0.9GB（FP8），计算量取决于 batch 大小和输出序列长度。

第6条　FFN（Feedforward Neural Network，前馈神经网络）

FFN 是 Transformer 中除注意力机制外的另一个核心组件，负责对每个位置的表示进行

非线性变换。

第7条　结构与计算

FFN 本质上是一种**多层感知机（MLP, Multi-Layer Perceptron）**，由两层线性变换和一个激活函数组成。其数学表达式为：

$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2\$$

其中：

1) $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$和$W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$为权重矩阵
2) b_1和b_2为偏置向量
3) $\max(0,\cdot)$为ReLU激活函数
4) d_{model}为模型维度（通常为512），d_{ff}为隐藏层维度（通常为2048）

   FFN对序列中每个位置的表示独立进行变换，引入非线性（ReLU）并允许模型学习更深层次的表示变换。与注意力机制关注序列内不同位置间的关系不同，FFN专注于对单个位置的表示进行深度变换。

第8条　FFN与MLP的关系

FFN（Feedforward Neural Network）是前馈神经网络的简称，其结构与**MLP（Multi-Layer Perceptron，多层感知机）**相同，均由多层线性变换和激活函数组成。在 Transformer 中，FFN 特指两层线性变换中间夹一个 ReLU 激活函数的结构，可以视为 MLP 的一种特定实现形式。因此，FFN 和 MLP 在概念上可以互换使用，FFN 是 MLP 在 Transformer 架构中的具体应用。

第9条　FFN变体

1) **Dense MLP（稠密MLP）**标准的全连接前馈网络，每个输入token激活所有参数。在DeepSeekMoE的研究中，稠密模型（每个token激活所有参数）被用作MoE模型性能的理论上限参考。
2) **MoE（Mixture of Experts，混合专家模型）**MoE将FFN划分为多个子网络（专家），每个输入token仅激活部分专家，从而实现条件计算。MoE层包含两个关键组件：
   a) **门控网络（Gating Network）**：决定每个token应激活哪些专家（通常采用TopK门控策略）
   b) **专家网络（Expert Network）**：多个独立的FFN子网络，每个专家学习不同的知识表示

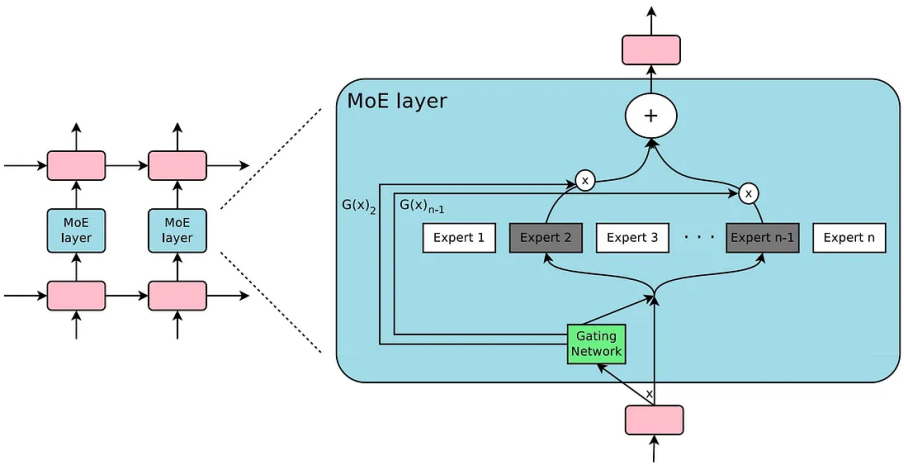


图 2.4 Google 的 MoE

传统的 TopK MoE 存在两个问题：(1) **知识混合性**：有限数量的专家需要处理多样化的知识，导致专家难以专精；(2) **知识冗余**：不同专家可能学习到相同的共享知识，造成参数冗余。

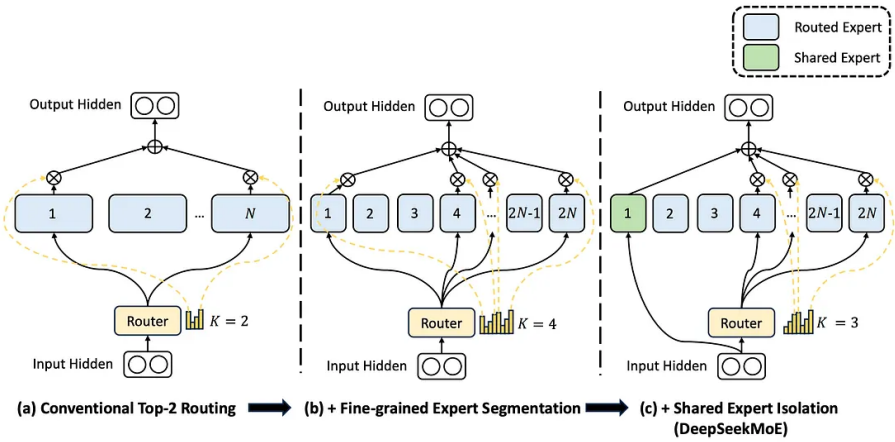


图 2.5 DeepSeek 的 MoE

DeepSeekMoE 通过细分更多专家并引入共享专家机制，缓解了上述问题，使 MoE 模型更接近理论性能上限。

3) **DeepSeekMoE（DeepSeek Mixture of Experts）** DeepSeekMoE是DeepSeek团队提出的改进型MoE架构（参考DeepSeekMoE论文），通过两个核心策略实现"最终专家专精"：

1. 细粒度专家分割与激活

将N#个专家细分为mN个小专家，从中激活mK个：

激活率 $= \frac{mK}{mN} = \frac{K}{N}$$

虽然激活率保持不变，但mN个小专家（每个维度为d_{ff}/m）比N#个大

专家（维度为d_{ff}）更容易保持差异化，减少知识冗余，提高表达能力。

2. 共享专家隔离机制

预留K_s个共享专家，所有token都通过：

$$\text{output} = \text{shared\_expert}(x) + \sum_{i=1}^{K} \text{routed\_expert}_i(x)$$

共享专家捕捉通用知识，路由专家处理任务特定知识，有效减少冗余。

计算复杂度

每token的FLOPs：

$$\text{FLOPs}_{\text{token}} = 2d \cdot d_{ff} \cdot \frac{mK + K_s}{mN} = 2d \cdot d_{ff} \cdot \frac{K + K_s/m}{N}$$

总参数量不增加（参考论文Section 3.1）：

$$\text{Params}_{\text{total}} = \text{共享专家参数} + N \times (2d \cdot d_{ff})$$