模型详细配置

《大语言模型》编写团队：李军毅

大语言模型的参数配置

AI Box

- 构建大模型需要考虑的因素
  - 归一化方法
  - 位置编码
  - 激活函数
  - 注意力计算

层数L、注意力头数N、特征维度H

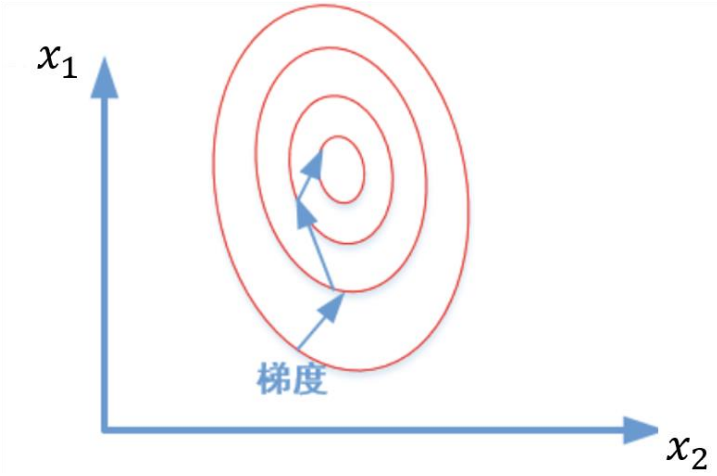
（根据模型规模大小确定）

模型	类别	大小	归一化	位置编码	激活函数	L	N	H
GPT-3	因果	175B	Pre Layer	Learned	GELU	96	96	12288
PanGU-α	因果	207B	Pre Layer	Learned	GELU	64	128	16384
OPT	因果	175B	Pre Layer	Learned	ReLU	96	96	12288
PaLM	因果	540B	Pre Layer	RoPE	SwiGLU	118	48	18432
BLOOM	因果	176B	Pre Layer	ALiBi	GELU	70	112	14336
MT-NLG	因果	530B	-	-	-	105	128	20480
Gopher	因果	280B	Pre RMS	Relative	-	80	128	16384
Chinchilla	因果	70B	Pre RMS	Relative	-	80	64	8192
Galactica	因果	120B	Pre Layer	Learned	GELU	96	80	10240
LaMDA	因果	137B	-	Relative	GeGLU	64	128	8192
Jurassic-1	因果	178B	Pre Layer	Learned	GELU	76	96	13824
LLaMA-2	因果	70B	Pre RMS	RoPE	SwiGLU	80	64	8192
Pythia	因果	12B	Pre Layer	RoPE	GELU	36	40	5120
Baichuan-2	因果	13B	Pre RMS	ALiBi	SwiGLU	40	40	5120
Qwen-1.5	因果	72B	Pre RMS	RoPE	SwiGLU	80	64	8192
InternLM-2	因果	20B	Pre RMS	RoPE	SwiGLU	48	48	6144
Falcon	因果	180B	Pre Layer	RoPE	GELU	80	232	14848
MPT	因果	30B	Pre Layer	ALiBi	GELU	48	64	7168
Mistral	因果	7B	Pre RMS	RoPE	SwiGLU	32	32	4096
Gemma	因果	7B	Pre RMS	RoPE	GELU	28	16	3072
DeepSeek	因果	67B	Pre RMS	RoPE	SwiGLU	95	64	8192
Yi	因果	34B	Pre RMS	RoPE	SwiGLU	60	56	7168
YuLan	因果	12B	Pre RMS	RoPE	SwiGLU	40	38	4864
GLM-130B	前缀	130B	Post Deep	RoPE	GeGLU	70	96	12288
T5	编-解	11B	Pre RMS	Relative	ReLU	24	128	1024

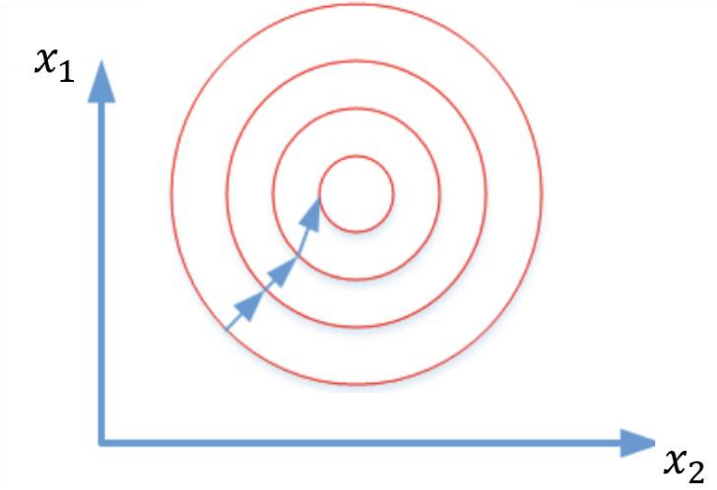
详细配置

AI Box

- 归一化
  - 为什么要做归一化?
    - 不同特征在空间中的尺度不同，对损失优化的影响不一致
    - 提升训练稳定性，加速模型收敛



(a) 未归一化数据的梯度下降过程



(b) 归一化数据的梯度下降过程

详细配置

AI Box

➢ 归一化

归一化方法	公式	基本思想	代表模型
层归一化 (LayerNorm)	$\text{LayerNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x}-\mu}{\sigma}\cdot\boldsymbol{\gamma}+\boldsymbol{\beta},$ $\mu = \frac{1}{H}\sum_{i=1}^{H} x_i, \quad \sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i-\mu))^2}.$	逐层计算所有激活值的均值和方差	GPT-3 BLOOM Pythia
均方根层归一化 (RMSNorm)	$\text{RMSNorm}(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\text{RMS}(\boldsymbol{x})}\cdot\boldsymbol{\gamma},$ $\text{RMS}(\boldsymbol{x}) = \sqrt{\frac{1}{H}\sum_{i=1}^{H} x_i^2}.$	只使用激活值总和的均方根进行放缩 训练速度更快，广泛使用	Gopher Chinchilla LLaMA DeepSeek
深度归一化 (DeepNorm)	$\text{DeepNorm}(\boldsymbol{x}) = \text{LayerNorm}(\alpha\cdot\boldsymbol{x}+\text{Sublayer}(\boldsymbol{x})),$	在残差连接中对激活值按比例α放缩	GLM-130B

详细配置

AI Box

- 归一化方法
  - RMSNorm代码实现

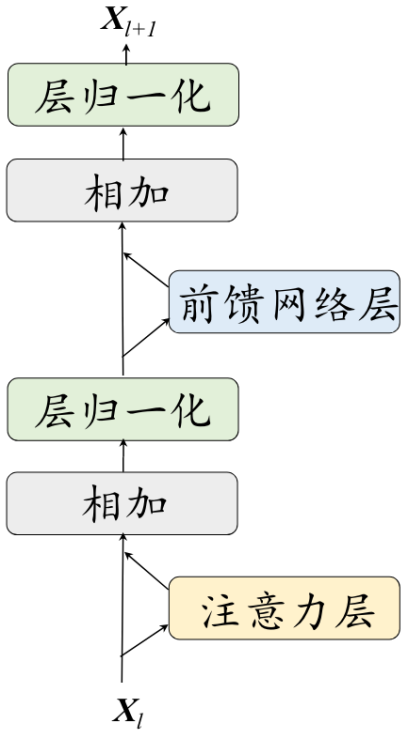
```python
class LlamaRMSNorm(nn.Module):
    def __init__(self, hidden_size, eps=1e-6):
        super().__init__()
        self.weight = nn.Parameter(torch.ones(hidden_size))
        self.variance_epsilon = eps

    def forward(self, hidden_states):
        input_dtype = hidden_states.dtype
        hidden_states = hidden_states.to(torch.float32)
        # 计算隐含状态的均方根
        variance = hidden_states.pow(2).mean(-1, keepdim=True)
        # 将隐含状态除以其均方根后重新缩放
        hidden_states = hidden_states * torch.rsqrt(variance +
        ↪  self.variance_epsilon)
        return self.weight * hidden_states.to(input_dtype)
```

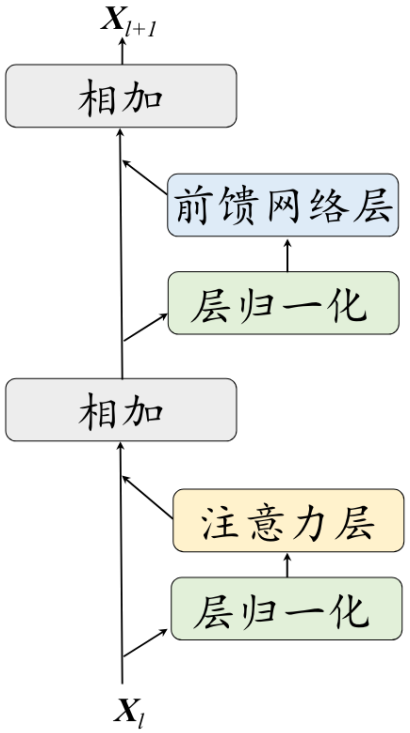
详细配置

AI Box

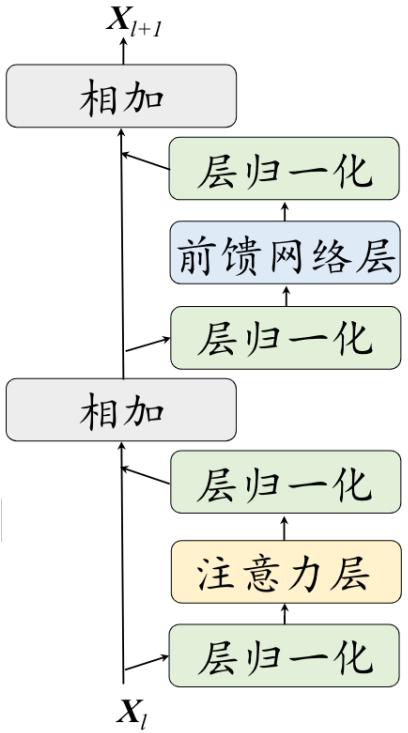
- 归一化模块位置
  - 层后归一化、层前归一化、夹心归一化



层后归一化



层前归一化



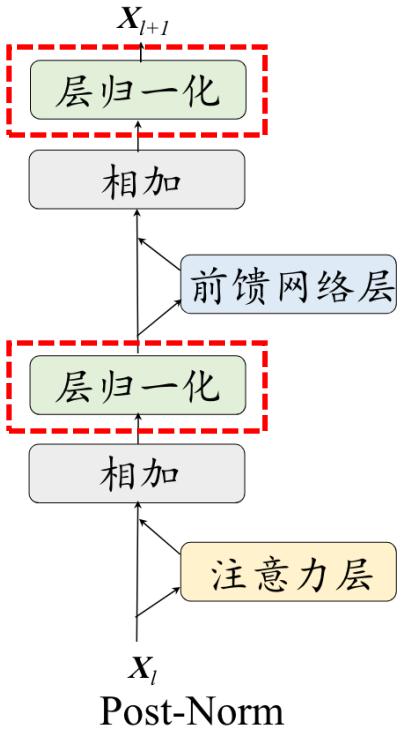
夹心归一化

详细配置

- 归一化模块位置
  - 层后归一化 (Post-Layer Normalization, Post-Norm)
    - 归一化模块放置于残差计算之后

$$\text{Post-Norm}(\boldsymbol{x}) = \text{Norm}(\boldsymbol{x} + \text{Sublayer}(\boldsymbol{x})),$$

    - 优点：加快训练收敛速度，防止梯度爆炸或梯度消失，降低神经网络对于超参数的敏感性
    - 缺点：可能导致训练不稳定，目前较少单独使用

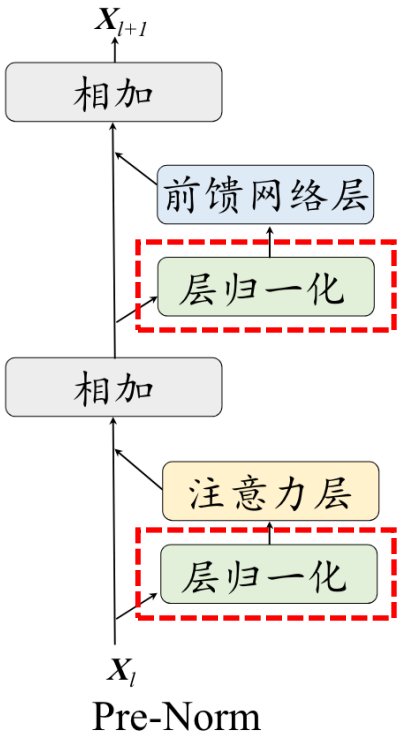


详细配置

- 归一化模块位置
  - 层前归一化 (Pre-Layer Normalization, Pre-Norm)
    - 归一化模块应用在每个子层之前

$$\text{Pre-Norm}(\boldsymbol{x}) = \boldsymbol{x} + \text{Sublayer}(\text{Norm}(\boldsymbol{x})),$$

    - 缺点：性能略有逊色
    - 优点：训练更加稳定，主流模型采用较多



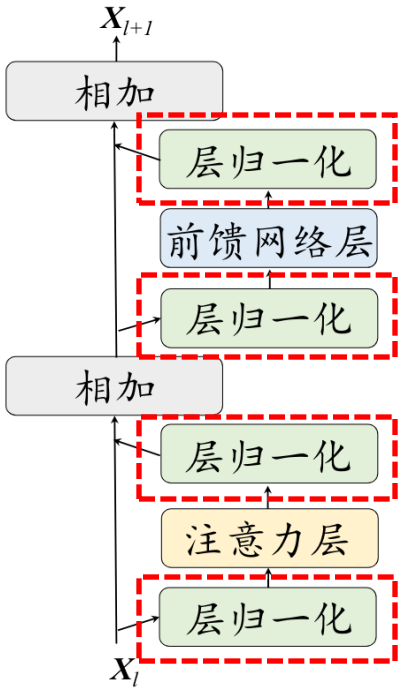
Pre-Norm

详细配置

- 归一化模块位置
  - 夹心归一化 (Sandwich-Norm)
    - 可以看作是 Pre-Norm 和 Post-Norm 两种方法的组合

$$\text{Sandwich-Norm}(\boldsymbol{x}) = \boldsymbol{x} + \text{Norm}(\text{Sublayer}(\text{Norm}(\boldsymbol{x}))).$$

    - 理论上更优，但仍无法保证稳定训练



Sandwich-Norm

详细配置

AI Box

➢ 激活函数

激活函数	公式	代表模型
ReLU	$\mathrm{ReLU}(x) = \max(x, 0)$	OPT、T5
Swish	$\mathrm{Swish}(x) = x \cdot \mathrm{sigmoid}(x)$	PaLM、Qwen-1.5、DeepSeek
SwiGLU	$\mathrm{SwiGLU}(\boldsymbol{x}) = \mathrm{Swish}(\boldsymbol{W}^G\boldsymbol{x}) \odot (\boldsymbol{W}^U\boldsymbol{x})$	
GELU	$\mathrm{GELU}(x) = 0.5x \cdot [1 + \mathrm{erf}(x/\sqrt{2})]$ $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$	GPT-3、PanGU-α、Pythia BLOOM、Falcon
GEGLU	$\mathrm{GeGLU}(\boldsymbol{x}) = \mathrm{GELU}(\boldsymbol{W}^G\boldsymbol{x}) \odot (\boldsymbol{W}^U\boldsymbol{x})$	LaMDA、GLM-130B

详细配置

AI Box

➢ 位置编码

绝对位置编码注意力计算

$$
\begin{aligned}
A_{ij} &= \boldsymbol{x}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{x}_j^{\mathsf{T}} \\
&= (\boldsymbol{v}_i + \boldsymbol{p}_i) \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} (\boldsymbol{v}_j + \boldsymbol{p}_j)^{\mathsf{T}} \\
&= \boldsymbol{v}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{v}_j^{\mathsf{T}} + \boldsymbol{v}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{p}_j^{\mathsf{T}} + \boldsymbol{p}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{v}_j^{\mathsf{T}} + \boldsymbol{p}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{p}_j^{\mathsf{T}},
\end{aligned}
$$

查询词嵌入和键词嵌入相乘　查询词嵌入和键位置编码相乘　查询位置编码和键词嵌入相乘　查询位置编码和键位置编码相乘

相对位置编码注意力计算

$$
A_{ij} = \boldsymbol{v}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{v}_j^{\mathsf{T}} + \boldsymbol{v}_i \boldsymbol{W}^Q \boldsymbol{W}^{R\mathsf{T}} \boldsymbol{r}_{i-j}^{\mathsf{T}} + f \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{v}_j^{\mathsf{T}} + g \boldsymbol{W}^{R\mathsf{T}} \boldsymbol{r}_{i-j}^{\mathsf{T}},
$$

键位置编码替换为相对位置编码 $\boldsymbol{r}_{i-j}\boldsymbol{W}^R$　查询位置编码替换为全局参数 f, g　键位置编码替换为相对位置编码 $\boldsymbol{r}_{i-j}\boldsymbol{W}^R$

详细配置

AI Box

- 位置编码
  - 旋转位置编码 (Rotary Position Embedding, RoPE)
    - 使用了基于绝对位置信息的旋转矩阵来表示注意力中的相对位置信息
    - 为序列中每个绝对位置设置了特定的旋转矩阵 $\boldsymbol{R}_{\theta,t}$ (位置索引为 t)

$$\boldsymbol{R}_{\theta,t} = \begin{bmatrix} \cos t\theta_1 & -\sin t\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin t\theta_1 & \cos t\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos t\theta_2 & -\sin t\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin t\theta_2 & \cos t\theta_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos t\theta_{H/2} & -\sin t\theta_{H/2} \\ 0 & 0 & 0 & 0 & \dots & \sin t\theta_{H/2} & \cos t\theta_{H/2} \end{bmatrix}$$

详细配置

- 位置编码
  - 旋转位置编码 (Rotary Position Embedding, RoPE)
    - 在处理query和key向量时，将连续出现的两个元素视为一个子空间
    - 每一个子空间 i 所对应的两个元素都会根据一个特定的旋转角度 $t \cdot \theta_i$ 进行旋转

$$\boldsymbol{R}_{\theta,t} = \begin{bmatrix} \cos t\theta_1 & -\sin t\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin t\theta_1 & \cos t\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos t\theta_2 & -\sin t\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin t\theta_2 & \cos t\theta_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos t\theta_{H/2} & -\sin t\theta_{H/2} \\ 0 & 0 & 0 & 0 & \dots & \sin t\theta_{H/2} & \cos t\theta_{H/2} \end{bmatrix}$$

子空间

RoPE 将旋转基 θ_i 定义为底数 b 的指数

$$\Theta = \{\theta_i = b^{-2(i-1)/H} | i \in \{1, 2, \dots, H/2\}\}$$

详细配置

AI Box

- 位置编码
  - 旋转位置编码 (Rotary Position Embedding, RoPE)
    - 根据三角函数的特性，位置索引为 i 的旋转矩阵与位置索引为 j 的旋转矩阵的乘积等于位置索引为相对距离 $i-j$ 的旋转矩阵，即

$$R_{\theta,i}R_{\theta,j}^{\mathsf{T}} = R_{\theta,i-j}$$

    - 通过这种方式将相对位置信息融入注意力分数

$$\boldsymbol{q}_i = \boldsymbol{x}_i \boldsymbol{W}^Q \boldsymbol{R}_{\theta,i},$$

$$\boldsymbol{k}_j = \boldsymbol{x}_j \boldsymbol{W}^K \boldsymbol{R}_{\theta,j},$$

$$A_{ij} = (\boldsymbol{x}_i \boldsymbol{W}^Q \boldsymbol{R}_{\theta,i})(\boldsymbol{x}_j \boldsymbol{W}^K \boldsymbol{R}_{\theta,j})^{\mathsf{T}} = \boldsymbol{x}_i \boldsymbol{W}^Q \boldsymbol{R}_{\theta,i-j} \boldsymbol{W}^{K\mathsf{T}} \boldsymbol{x}_j^{\mathsf{T}}.$$

详细配置

AI Box

- 位置编码
  - 旋转位置编码代码实现

```python
def rotate_half(x):
    # 将向量每两个元素视为一个子空间
    x1 = x[..., : x.shape[-1] // 2]
    x2 = x[..., x.shape[-1] // 2 :]
    return torch.cat((-x2, x1), dim=-1)

def apply_rotary_pos_emb(q, k, cos, sin, position_ids):
    # 获得各个子空间旋转的正余弦值
    cos = cos[position_ids].unsqueeze(1)
    sin = sin[position_ids].unsqueeze(1)
    # 将每个子空间按照特定角度进行旋转
    q_embed = (q * cos) + (rotate_half(q) * sin)
    k_embed = (k * cos) + (rotate_half(k) * sin)
    return q_embed, k_embed
```

详细配置

AI Box

- 位置编码
  - ALiBi 位置编码
    - 用于增强 Transformer 模型的长度外推能力
    - 引入了与相对距离成比例关系的惩罚因子来调整注意力分数

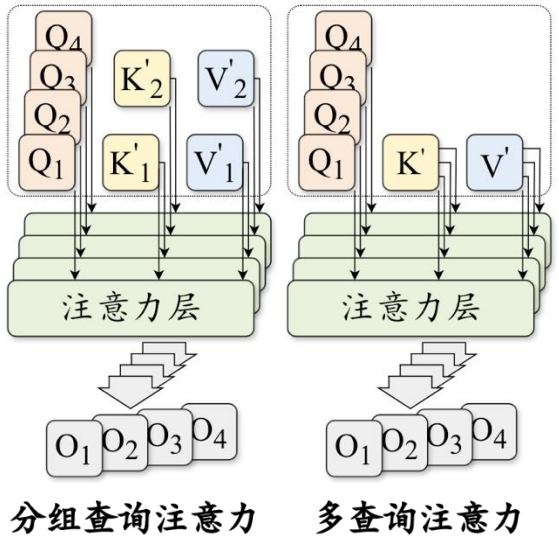
$$A_{ij} = \boldsymbol{x}_i \boldsymbol{W}^Q \boldsymbol{W}^{K\top} \boldsymbol{x}_j^\top - m(i-j),$$

    - $i-j$：查询和键之间的位置偏移量
    - m :每个注意力头独有的惩罚系数

详细配置

AI Box

➢ 注意力机制

注意力	思路	优缺点
完整注意力	每个词元关注所有前序词元	计算复杂度随长度平方级增长
多查询注意力 (Multi-Query Attention, MQA)	所有注意力头共享键值变换矩阵 代表模型如PaLM、StarCoder	减少访存量，提升计算速度 
分组查询注意力 (Grouped-Query Attention, GQA)	同一组内的注意力头共享键值变换矩阵 代表模型如LLaMA-2	

详细配置

AI Box

- 多头隐式注意力 (Multi-Head Latent Attention, MLA)
  - 由DeepSeek-V2 提出，主要目的是降低推理时KV Cache的存储开销

标准多头注意力

$$[\mathbf{q}_{t,1};\mathbf{q}_{t,2};\ldots;\mathbf{q}_{t,n_h}]=\mathbf{q}_t,$$
$$[\mathbf{k}_{t,1};\mathbf{k}_{t,2};\ldots;\mathbf{k}_{t,n_h}]=\mathbf{k}_t,$$
$$[\mathbf{v}_{t,1};\mathbf{v}_{t,2};\ldots;\mathbf{v}_{t,n_h}]=\mathbf{v}_t,$$

推理时KV Cache需要缓存$\boldsymbol{k}_t,\boldsymbol{v}_t$，

参数量为$2n_hd_hl$

多头隐式注意力

$$\mathbf{c}_t^{KV}=W^{DKV}\mathbf{h}_t,$$
$$\mathbf{k}_t^C=W^{UK}\mathbf{c}_t^{KV},$$
$$\mathbf{v}_t^C=W^{UV}\mathbf{c}_t^{KV},$$

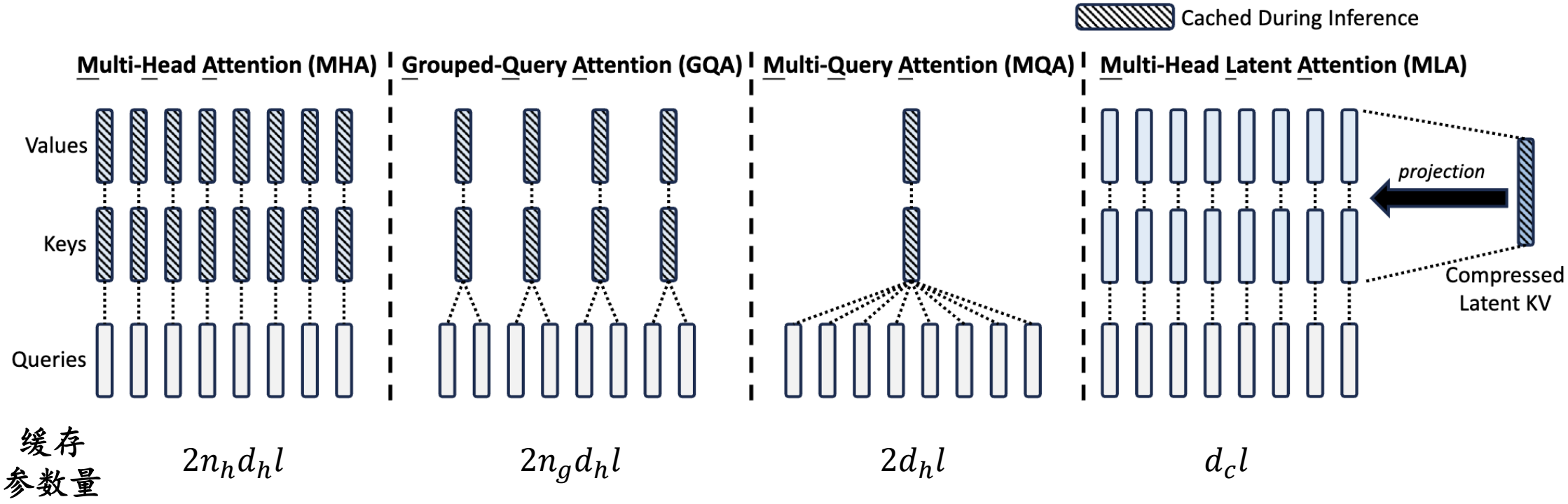
$\boldsymbol{c}_t^{KV}\in\mathbb{R}^{d_c}$：对key和value压缩后的隐向量 $(d_c\ll d_hn_h)$

推理时KV Cache只需要缓存$\boldsymbol{c}_t^{KV}$，参数量仅为d_cl

n_h为注意力头数目，d_h为每个头维度，l为层数，d_c为隐向量维度

详细配置

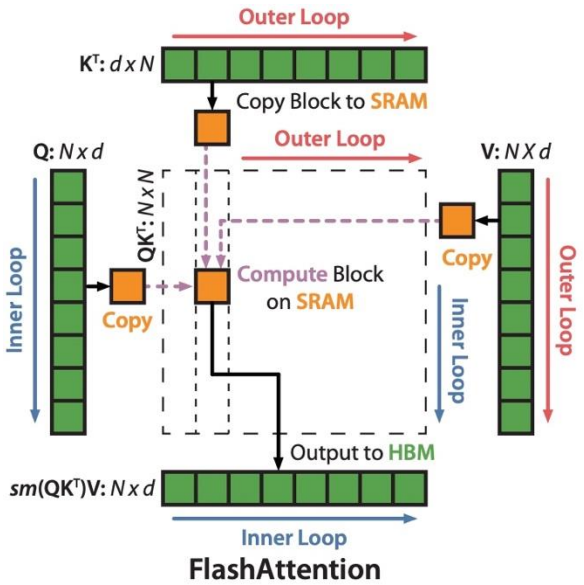
- 多头隐式注意力 (Multi-Head Latent Attention, MLA)
  - 由DeepSeek-V2 提出，主要目的是降低推理时KV Cache的存储开销

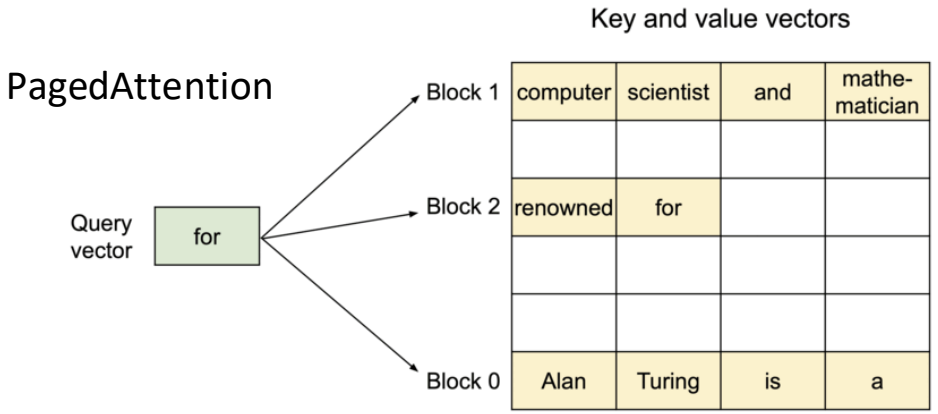


n_h为注意力头数目，d_h为每个头维度，l为层数，n_g为组的数目，d_c为隐向量维度

详细配置

- 注意力机制
  - 硬件优化的注意力机制
    - FlashAttention：通过矩阵分块计算以及减少内存读写次数提高了计算效率
    - PagedAttention：针对解码阶段，对KV Cache进行分块存储并优化计算方式

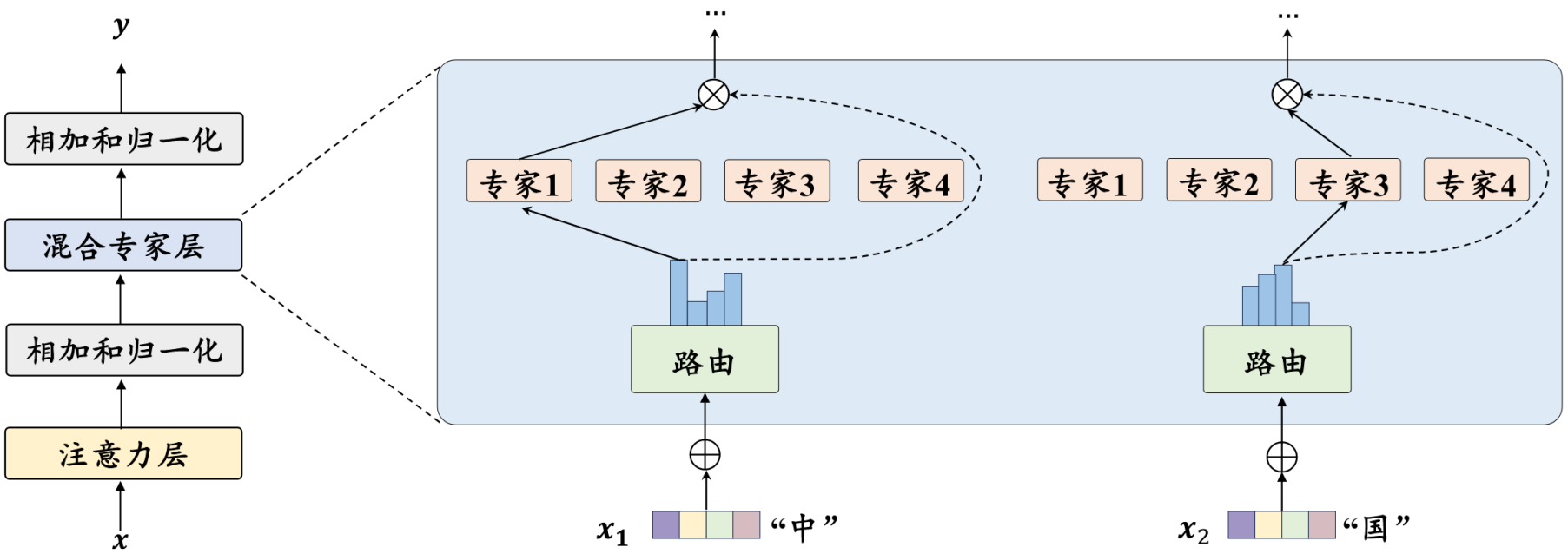




FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. **NeurIPS 2022**.

详细配置

- 混合专家架构 (Mixture-of-Experts, MoE)
  - 例如 DeepSeek-V2/V3/R1、Mixtral等
  - 旨在不显著提升计算成本的同时实现对于模型参数的拓展



详细配置

AI Box

- 混合专家架构 (Mixture-of-Experts, MoE)
  - 包含 K 个由前馈神经网络构成的专家组件 E_i
  - 通过路由网络 G 计算词元 x_t 对应于各个专家的权重

$$G(\boldsymbol{x}_t) = \text{softmax}(\text{topk}(\boldsymbol{x}_t \cdot W^G))$$

    - W^G：将词元映射为 K 个专家的得分
    - topk：选择出得分最高的 k 个专家进行激活
    - softmax：计算专家权重，未被选择的专家权重被置为0
  - 被选择专家的输出加权和，作为该混合专家网络层的最终输出 o_t

$$\boldsymbol{o}_t = \text{MoELayer}(\boldsymbol{x}_t) = \sum_{i=1}^{K} G(\boldsymbol{x}_t)_i \cdot E_i(\boldsymbol{x}_t).$$

LLaMA与DeepSeek模型配置比较

AI Box

配置	LLaMA-3.1 (405B)	DeepSeek (67B)	DeepSeek-V2 (236B)	DeepSeek-V3 (671B)
混合专家	N/A	N/A	162 Experts	257 Experts
归一化	Pre RMSNorm	Pre RMSNorm	Pre RMSNorm	Pre RMSNorm
位置编码	旋转位置编码	旋转位置编码	旋转位置编码	旋转位置编码
激活函数	SwiGLU	SwiGLU	SwiGLU	SwiGLU
注意力	分组查询注意力	分组查询注意力	多头隐式注意力	多头隐式注意力
层数	126	95	60	61
隐藏层维度	16384	8192	5120	7168
注意力头个数	128	64	128	128

谢谢