

缓存与效果的极限拉扯：从MHA、MQA、GQA到MLA

原文地址：<https://spaces.ac.cn/archives/10091> 本文为其修改精简版本

关键概念：

多头注意力（Multi-Head Attention, MHA）

分组查询注意力（Grouped Query Attention, GQA）

多查询注意力（Multi-Query Attention, MQA）

多头潜变量注意力（Multi-Head Latent Attention, MLA）

KV Cache

内容目录：

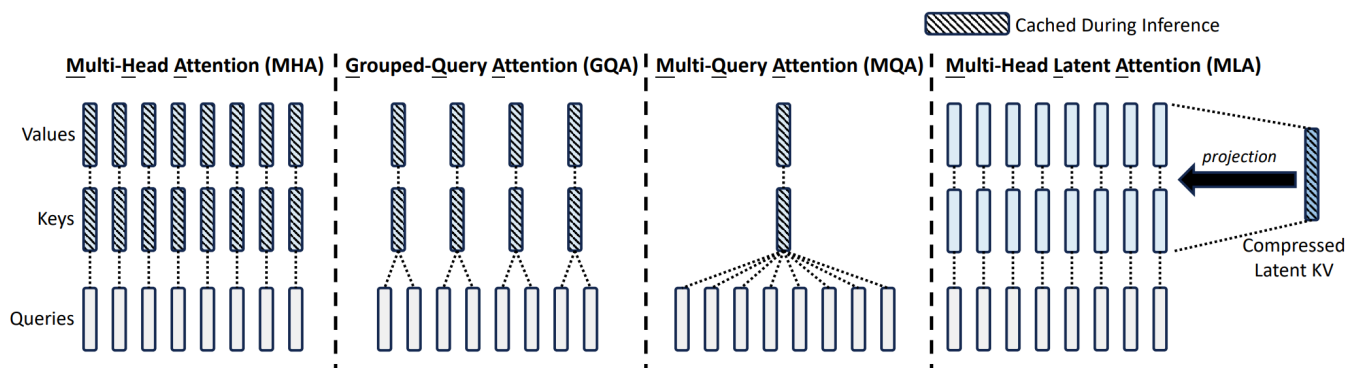
🎯 MHA（Multi-Head Attention）

🔧 MQA（Multi-Query Attention）

📁 GQA（Grouped-Query Attention）

🌀 MLA（Multi-head Latent Attention）

总览：本文概述了四种注意力机制——多头注意力（MHA）、分组查询注意力（GQA）、多查询注意力（MQA）以及多头潜变量注意力（MLA）。它们体现了注意力结构在**推理效率优化**上的演进路径：从最初的多头并行设计，到逐步减少键值存储的冗余，再到利用潜变量表示进一步压缩存储开销。特别是 MLA，通过将键（Key）和值（Value）联合编码为紧凑的潜在向量，在推理阶段显著降低了 KV Cache 缓存大小，为大规模模型的高效部署开辟了新方向。下图给出了一个直观的简化示意。



MHA (Multi-Head Attention)

MHA (Multi-Head Attention) 等价于多个独立的单头注意力拼接而成。假设输入的向量序列为 $\{x_1, x_2, \dots, x_l\}$ ，其中 $x_i \in \mathbb{R}^d$ 。在主流自回归 LLM 所用的 **Causal Attention** 中，MHA 可形式化记为：

$$\mathbf{o}_t = [\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)}]$$

$$\mathbf{o}_t^{(s)} = \text{Attention} \left(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)} \right) \triangleq \frac{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right) \mathbf{v}_i^{(s)}}{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right)}$$

$$\mathbf{q}_i^{(s)} = \mathbf{x}_i \mathbf{W}_q^{(s)} \in \mathbb{R}^{d_k}, \quad \mathbf{W}_q^{(s)} \in \mathbb{R}^{d \times d_k}$$

$$\mathbf{k}_i^{(s)} = \mathbf{x}_i \mathbf{W}_k^{(s)} \in \mathbb{R}^{d_k}, \quad \mathbf{W}_k^{(s)} \in \mathbb{R}^{d \times d_k}$$

$$\mathbf{v}_i^{(s)} = \mathbf{x}_i \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_v}, \quad \mathbf{W}_v^{(s)} \in \mathbb{R}^{d \times d_v}$$

KV Cache 在 token-by-token 递归生成时，新预测出来的第 $t + 1$ 个 token 并不会影响到已计算的 $\mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)}$ ，因此这部分结果可以缓存下来，供后续生成调用，从而避免不必要的重复计算。

然而，在长上下文推理中，KV Cache 的存储量会随着上下文长度和注意力头数线性增长，对显存和内存造成巨大压力。这种高占用会限制推理批量（batch size）和可处理的上下文长度，因此**减少 KV Cache 的开销**成为优化推理效率的重要方向。减少 KV Cache 的主要目标是：

- 在更少的设备上推理更长的上下文（Context）；
- 在相同 Context 长度下支持更大的推理 batch size。

这不仅可以加快推理速度、提升吞吐量，还能显著降低推理成本。

MQA（Multi-Query Attention）

论文地址：[Fast Transformer Decoding: One Write-Head is All You Need](#)

代表模型：[PaLM](#)、[StarCoder](#)、[Gemini](#)

在多头注意力（MHA）中，每个 Attention Head 都会维护自己独立的键（Key）和值（Value）序列，因此 KV Cache 的存储量会随着注意力头数 h 成倍增加。这在长上下文推理中会带来非常高的显存开销。

MQA (Multi-Query Attention) 的核心思想是**让所有 Attention Head 共享同一个 K、V**，即取消 MHA 中所有 k, v 的上标 (s) ：

$$\mathbf{o}_t = [\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)}]$$
$$\mathbf{o}_t^{(s)} = \text{Attention}(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)}) \triangleq \frac{\sum_{i \leq t} \exp(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top}) \mathbf{v}_i^{(s)}}{\sum_{i \leq t} \exp(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top})}$$
$$\begin{aligned} \mathbf{q}_i^{(s)} &= \mathbf{x}_i \mathbf{W}_q^{(s)} \in \mathbb{R}^{d_k}, & \mathbf{W}_q^{(s)} &\in \mathbb{R}^{d \times d_k} \\ \mathbf{k}_i^{(s)} &= \mathbf{x}_i \mathbf{W}_k^{(s)} \in \mathbb{R}^{d_k}, & \mathbf{W}_k^{(s)} &\in \mathbb{R}^{d \times d_k} \\ \mathbf{v}_i^{(s)} &= \mathbf{x}_i \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_v}, & \mathbf{W}_v^{(s)} &\in \mathbb{R}^{d \times d_v} \end{aligned}$$

这样一来，**KV Cache 的大小直接减少到原来的 $1/h$** ，显存占用得到了极大优化——从节省空间的角度来看几乎接近“天花板级别”的压缩效果。

在性能方面，目前研究表明 MQA 在大多数任务上的精度损失有限，且支持者认为这部分差距可以通过额外训练来弥补。此外，由于共享 K、V，Attention 层的参数量减少近一半，因此在保持模型总参数量不变的前提下，通常会增大 FFN/GLU 的规模，这也能部分抵消精度下降的影响。

GQA（Grouped-Query Attention）

论文地址：[GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#)

代表模型：Meta 开源的 [LLAMA2-70B](#) 以及 [LLAMA3](#) 全系列，[TigerBot](#)、[DeepSeek-V1](#)、[StarCoder2](#)、[Yi](#)、[ChatGLM2](#)、[ChatGLM3](#) 等。

在多头注意力（MHA）中，所有 h 个注意力头都各自存储一份键（Key）和值（Value），这会让 KV Cache 占用显存的规模与 h 成正比。虽然 MQA 通过让所有 Head 共享同一个 K、V 把显存压缩到了极致，但可能带来一定精度下降。

GQA (Grouped-Query Attention) 设计了一种折中方案：将 h 个 Attention Head 划分为 g 个组（ g 可整除 h ），每组共享同一对 K、V。数学表示为：

$$\mathbf{o}_t = [\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)}]$$

$$\mathbf{o}_t^{(s)} = \text{Attention} \left(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(\lceil sg/h \rceil)}, \mathbf{v}_{\leq t}^{(\lceil sg/h \rceil)} \right) \triangleq \frac{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(\lceil sg/h \rceil)\top} \right) \mathbf{v}_i^{(\lceil sg/h \rceil)}}{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(\lceil sg/h \rceil)\top} \right)}$$

$$\begin{aligned} \mathbf{q}_i^{(s)} &= \mathbf{x}_i \mathbf{W}_q^{(s)} \in \mathbb{R}^{d_k}, & \mathbf{W}_q^{(s)} &\in \mathbb{R}^{d \times d_k} \\ \mathbf{k}_i^{(\lceil sg/h \rceil)} &= \mathbf{x}_i \mathbf{W}_k^{(\lceil sg/h \rceil)} \in \mathbb{R}^{d_k}, & \mathbf{W}_k^{(\lceil sg/h \rceil)} &\in \mathbb{R}^{d \times d_k} \\ \mathbf{v}_i^{(\lceil sg/h \rceil)} &= \mathbf{x}_i \mathbf{W}_v^{(\lceil sg/h \rceil)} \in \mathbb{R}^{d_v}, & \mathbf{W}_v^{(\lceil sg/h \rceil)} &\in \mathbb{R}^{d \times d_v} \end{aligned}$$

这里 $\lceil \cdot \rceil$ 表示上取整。当 $g = h$ 时，GQA 等价于 MHA；当 $g = 1$ 时，就完全变成了 MQA；而当 $1 < g < h$ 时，KV Cache 大小为原来的 g/h 。

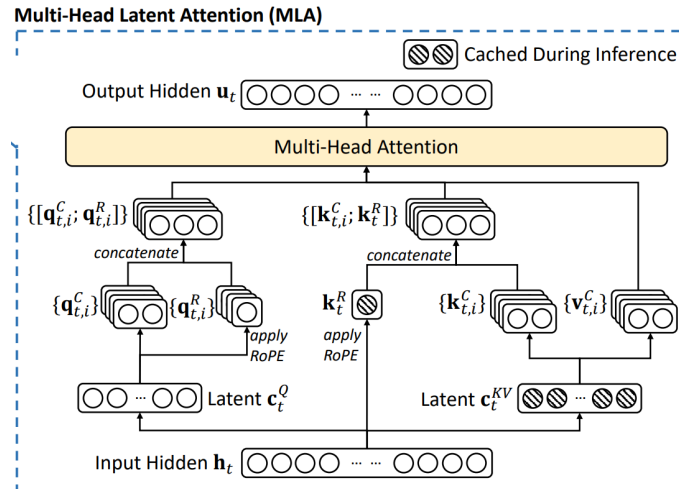
这种设计虽然压缩率不如 MQA，但提供了更大的自由度，可以在显存占用与性能损失之间找到更平衡的点，因此被很多大规模 LLM（如 LLaMA 系列）广泛采用。

MLA (Multi-head Latent Attention)

代表模型：DeepSeek-V2、DeepSeek-V3 等

在多头注意力中，即便采用了 GQA/MQA，推理阶段仍需缓存较高维度的 K、V 向量，占用大量显存。

MLA (Multi-head Latent Attention) 则提出通过潜变量表示进一步压缩 KV Cache，同时兼顾位置编码兼容性与训练显存优化。MLA 的框架直观示意图如下：



KV Cache 压缩 MLA 不直接缓存每个 Head 的 k_i, v_i ，而是通过低秩投影将它们联合映射为一个共享的潜向量 c_i （维度更低），从而减少缓存。这一过程对应框架图中右路，形式化如下：

$$\mathbf{o}_t = [\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)}]$$

$$\mathbf{o}_t^{(s)} = \text{Attention} \left(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)} \right) \triangleq \frac{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right) \mathbf{v}_i^{(s)}}{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right)}$$

$$\begin{aligned} \mathbf{q}_i^{(s)} &= \mathbf{x}_i \mathbf{W}_q^{(s)} \in \mathbb{R}^{d_k}, & \mathbf{W}_q^{(s)} &\in \mathbb{R}^{d \times d_k} \\ \mathbf{k}_i^{(s)} &= \mathbf{c}_i \mathbf{W}_k^{(s)} \in \mathbb{R}^{d_k}, & \mathbf{W}_k^{(s)} &\in \mathbb{R}^{d_c \times d_k} \\ \mathbf{v}_i^{(s)} &= \mathbf{c}_i \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_v}, & \mathbf{W}_v^{(s)} &\in \mathbb{R}^{d_c \times d_v} \end{aligned}$$

$$\mathbf{c}_i = \mathbf{x}_i \mathbf{W}_c \in \mathbb{R}^{d_c}, \quad \mathbf{W}_c \in \mathbb{R}^{d \times d_c},$$

对于推理阶段，此时需要利用缓存的 c_i 来进行推理，具体来说根据

$$\mathbf{q}_i^{(s)} \mathbf{k}_i^{(s)\top} = \left(\mathbf{x}_t \mathbf{W}_q^{(s)} \right) \left(c_i \mathbf{W}_k^{(s)} \right)^\top = \mathbf{x}_t \left(\mathbf{W}_q^{(s)} \mathbf{W}_k^{(s)\top} \right) c_i^\top$$

并将合并的权值矩阵作为Q的新投影矩阵来实现。同理，也可以对 v_i 进行处理。

兼容 RoPE (旋转位置编码) 由于低秩压缩不兼容 RoPE (旋转位置编码)，原因简要解释如下：由于之前只缓存了 c_i 来替代 k_i ，这也使得没有缓存 $W_k^{(s)}$ ；根据 RoPE 的计算过程来看，旋转位置编码以一个正交矩阵右乘的方式来作用在 $k_i^{(s)}$ 上，因此此时缺乏 $W_k^{(s)}$ 无法做到这一点；直观上来看，这个问题也可以理解为此时维度混合成低维表示，RoPE 的逐维相位信息已丢失。为了应对这个挑战，MLA 采用了一种折中方案：

- 在每个 Head 的 Q、K 向量中额外增加 d_r 维专门用于 RoPE；
- 其中 K 的 RoPE 维度在各 Head 间共享，从而类似 MQA 节约显存；

这一过程对应框架图中的中路，主要目的是加入附加维度补偿位置信息，更精确的数学表达式如下：

$$\begin{aligned} \mathbf{o}_t &= \left[\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)} \right] \\ \mathbf{o}_t^{(s)} &= \text{Attention} \left(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)} \right) \triangleq \frac{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right) \mathbf{v}_i^{(s)}}{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right)} \\ \mathbf{q}_i^{(s)} &= \left[\mathbf{x}_i \mathbf{W}_{qc}^{(s)}, \mathbf{x}_i \mathbf{W}_{qr}^{(s)} \mathcal{R}_i \right] \in \mathbb{R}^{d_k + d_r}, \quad \mathbf{W}_{qc}^{(s)} \in \mathbb{R}^{d \times d_k}, \mathbf{W}_{qr}^{(s)} \in \mathbb{R}^{d \times d_r} \\ \mathbf{k}_i^{(s)} &= \left[c_i \mathbf{W}_{kc}^{(s)}, \mathbf{x}_i \mathbf{W}_{kr}^{(s)} \mathcal{R}_i \right] \in \mathbb{R}^{d_k + d_r}, \quad \mathbf{W}_{kc}^{(s)} \in \mathbb{R}^{d_c \times d_k}, \mathbf{W}_{kr}^{(s)} \in \mathbb{R}^{d \times d_r} \\ \mathbf{v}_i^{(s)} &= c_i \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_v}, \quad \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_c \times d_v} \\ c_i &= \mathbf{x}_i \mathbf{W}_c \in \mathbb{R}^{d_c}, \quad \mathbf{W}_c \in \mathbb{R}^{d \times d_c} \end{aligned}$$

其中 \mathcal{R}_i 为位置编码，注意如果 K 的附加部分不共享，此时缓存跟 MHA 类似因此起不到作用。

Query 低秩投影 为进一步减少训练期间的参数量和梯度显存占用，MLA 将 Q 的输入也改为低秩投影形式（与 KV Cache 压缩无关）。这部分对应框架图中的左路，数学形式化表达为：

$$\begin{aligned} \mathbf{o}_t &= \left[\mathbf{o}_t^{(1)}, \mathbf{o}_t^{(2)}, \dots, \mathbf{o}_t^{(h)} \right] \\ \mathbf{o}_t^{(s)} &= \text{Attention} \left(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{v}_{\leq t}^{(s)} \right) \triangleq \frac{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right) \mathbf{v}_i^{(s)}}{\sum_{i \leq t} \exp \left(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top} \right)} \\ \mathbf{q}_i^{(s)} &= \left[c'_i \mathbf{W}_{qc}^{(s)}, c'_i \mathbf{W}_{qr}^{(s)} \mathcal{R}_i \right] \in \mathbb{R}^{d_k + d_r}, \quad \mathbf{W}_{qc}^{(s)} \in \mathbb{R}^{d'_c \times d_k}, \mathbf{W}_{qr}^{(s)} \in \mathbb{R}^{d'_c \times d_r} \\ \mathbf{k}_i^{(s)} &= \left[c_i \mathbf{W}_{kc}^{(s)}, \mathbf{x}_i \mathbf{W}_{kr}^{(s)} \mathcal{R}_i \right] \in \mathbb{R}^{d_k + d_r}, \quad \mathbf{W}_{kc}^{(s)} \in \mathbb{R}^{d_c \times d_k}, \mathbf{W}_{kr}^{(s)} \in \mathbb{R}^{d \times d_r} \\ \mathbf{v}_i^{(s)} &= c_i \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_v}, \quad \mathbf{W}_v^{(s)} \in \mathbb{R}^{d_c \times d_v} \\ c'_i &= \mathbf{x}_i \mathbf{W}'_c \in \mathbb{R}^{d'_c}, \quad \mathbf{W}'_c \in \mathbb{R}^{d \times d'_c} \\ c_i &= \mathbf{x}_i \mathbf{W}_c \in \mathbb{R}^{d_c}, \quad \mathbf{W}_c \in \mathbb{R}^{d \times d_c} \end{aligned}$$

MLA 推理阶段总结 整个框架的推理阶段的计算过程重新表达成 MQA 的形式为

$$\mathbf{o}_t = [\mathbf{o}_t^{(1)} \mathbf{W}_v^{(1)}, \mathbf{o}_t^{(2)} \mathbf{W}_v^{(2)}, \dots, \mathbf{o}_t^{(h)} \mathbf{W}_v^{(h)}]$$

$$\mathbf{o}_t^{(s)} = \text{Attention}(\mathbf{q}_t^{(s)}, \mathbf{k}_{\leq t}^{(s)}, \mathbf{c}_{\leq t}) \triangleq \frac{\sum_{i \leq t} \exp(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top}) \mathbf{c}_i}{\sum_{i \leq t} \exp(\mathbf{q}_t^{(s)} \mathbf{k}_i^{(s)\top})}$$

$$\mathbf{q}_i^{(s)} = [\mathbf{c}'_i \mathbf{W}_{qc}^{(s)} \mathbf{W}_{kc}^{(s)\top}, \mathbf{c}'_i \mathbf{W}_{qr}^{(s)} \mathbf{R}_i] \in \mathbb{R}^{d_c + d_r}$$

$$\mathbf{k}_i^{(s)} = [\mathbf{c}_i, \mathbf{x}_i \mathbf{W}_{kr}^{(s)} \mathbf{R}_i] \in \mathbb{R}^{d_c + d_r}$$

$$\mathbf{W}_{qc}^{(s)} \in \mathbb{R}^{d'_c \times d_k}, \mathbf{W}_{kc}^{(s)} \in \mathbb{R}^{d_c \times d_k}, \mathbf{W}_{qr}^{(s)} \in \mathbb{R}^{d'_c \times d_r}, \mathbf{W}_{kr}^{(s)} \in \mathbb{R}^{d \times d_r}$$

$$\mathbf{c}'_i = \mathbf{x}_i \mathbf{W}'_c \in \mathbb{R}^{d'_c}, \quad \mathbf{W}'_c \in \mathbb{R}^{d \times d'_c}$$

$$\mathbf{c}_i = \mathbf{x}_i \mathbf{W}_c \in \mathbb{R}^{d_c}, \quad \mathbf{W}_c \in \mathbb{R}^{d \times d_c}$$

转换后，Q、K 的 Head Size 变为 $d_c + d_r$ ，V 的 Head Size 为 d_c （原论文设置下是 d_k, d_v 的 4 倍）。虽然这会增加一定的计算量，但推理阶段的大模型通常受带宽和显存限制更大，因此整体推理效率依然提升。

References

- [1] Liu A, Feng B, Wang B, et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model." arXiv preprint arXiv:2405.04434, 2024.
- [2] Liu A, Feng B, Xue B, et al. "Deepseek-v3 technical report". arXiv preprint arXiv:2412.19437, 2024.
- [3] N Shazeer et al. "Fast Transformer Decoding: One Write-Head is All You Need" arXiv preprint arXiv:1911.02150, 2019.
- [4] J Ainslie et al. "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints" arXiv preprint arXiv:2305.13245, 2023.