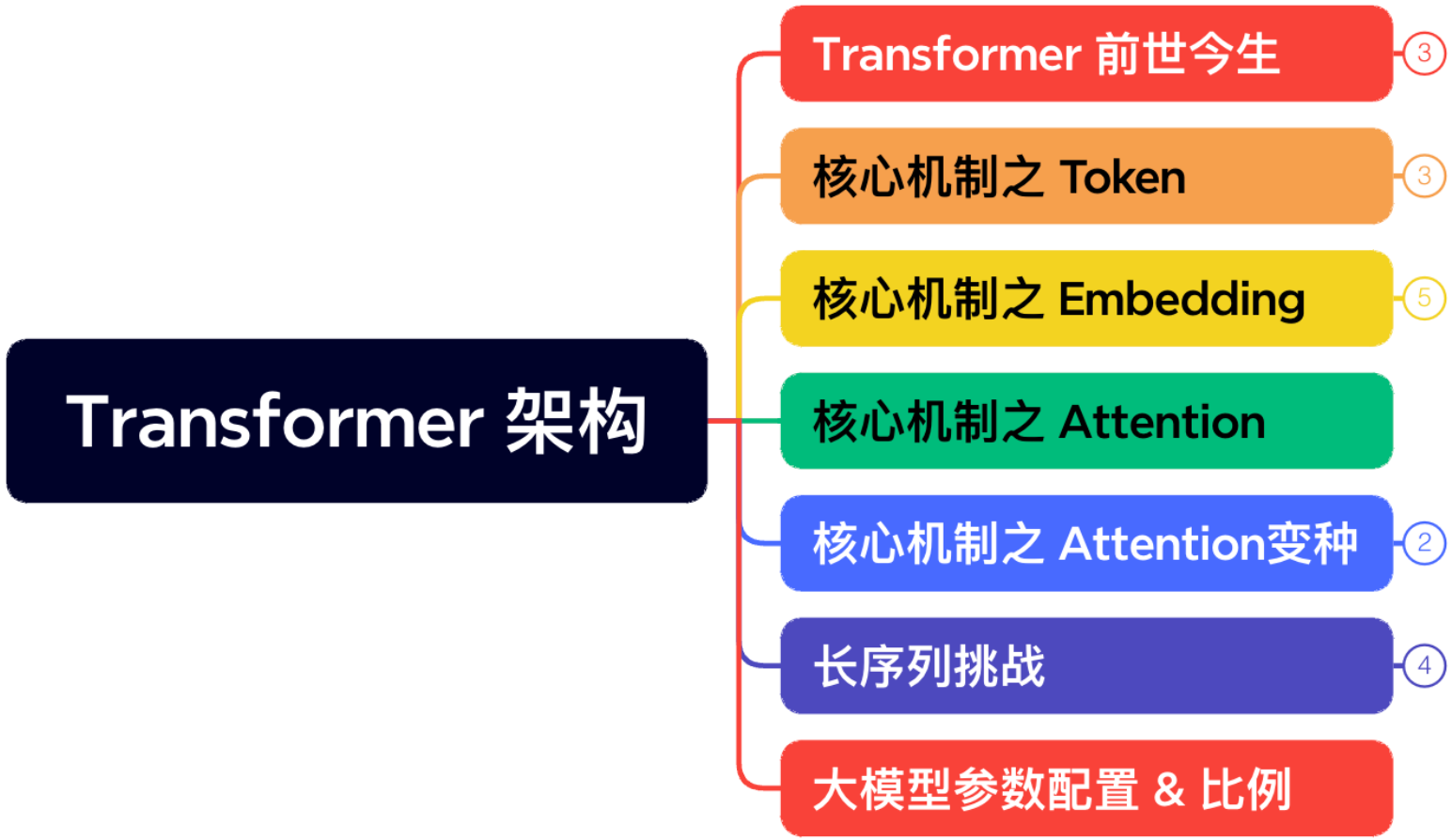



Attention 变种全解析


Contenet


Transformer 架构
Transformer 前世今生 3
核心机制之 Token 3
核心机制之 Embedding 5
核心机制之 Attention
核心机制之 Attention变种 2
长序列挑战 4
大模型参数配置 & 比例


好好想想

核心问题：

- 传统序列模型局限性——长程依赖丢失、并行计算困难。

解决方案：

- Attention核心思想——通过权重分配聚焦关键信息。

视频目录大纲

- Attention 挑战与优化方向
- 最新 Attention 架构演进（MQA、GQA、MLA）

01

Attention 挑战与优化方向

痛点分析

- **计算复杂度：** 传统Attention的 $O(n^2)$ 复杂度限制长序列处理。
- **内存瓶颈：**
  - 一般情况下LLM的推理都是在GPU上进行，单张GPU的显存是有限的，一部分要用来存放模型的参数和前向计算的激活值，这部分依赖于模型的体量，选定模型后它就是个常数。
  - 一部分要用来存放模型的KV Cache，这部分不仅依赖于模型的体量，还依赖于模型的输入长度，也就是在推理过程中是动态增长的，当Context长度足够长时，它的大小就会占主导地位

优化思路

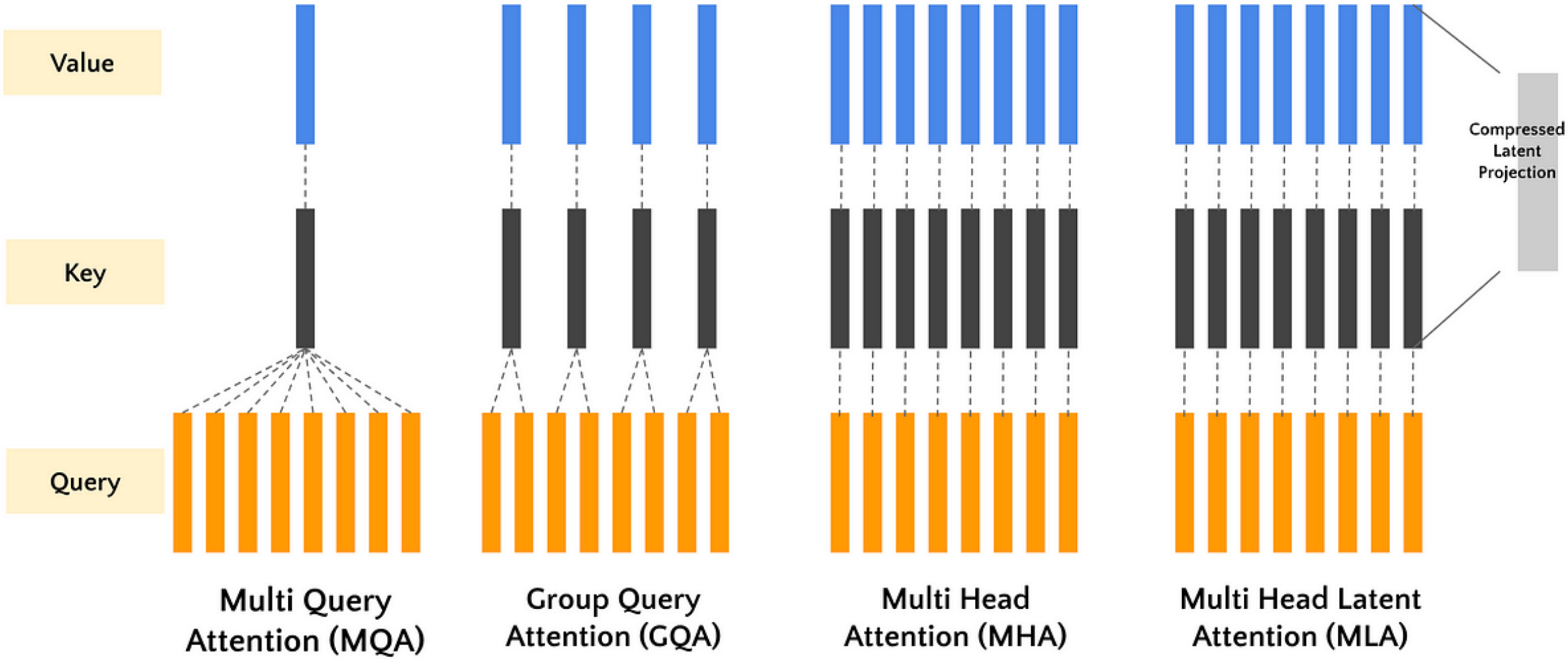
- 减少KV Cache的目的就是要实现在更少的设备上推理更长的Context，或者在相同的Context长度下让推理的batch size更大，从而实现更快的推理速度或者更大的吞吐总量。

- MQA、GQA、MLA，都是围绕“如何减少KV Cache同时尽可能地保证效果”这个主题发展而来的产物。

02

最新 Attention 架构演进

MQA、GQA、MHA、MLA



03

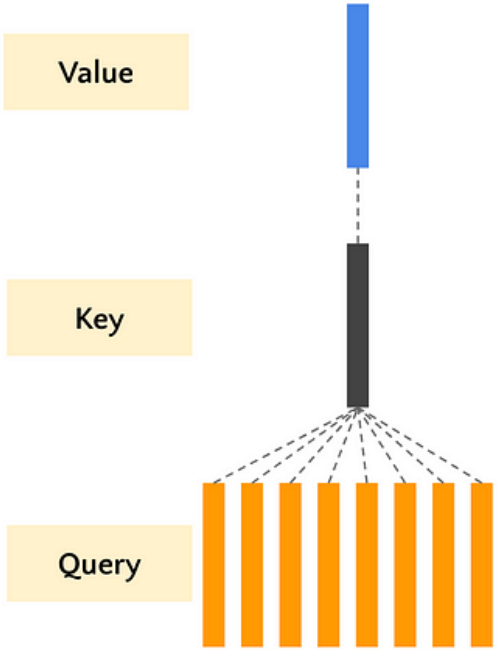
MQA

MQA： Multi-Query Attention

- 2019 年 Google 论文《Fast Transformer Decoding: One Write-Head is All You Need》, 每个 head 计算 attention 时，每个 head 的 K、V 都共享，只有 Q 在不同 head 是不同。

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

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



MQA： Multi-Query Attention

- KV Cache 只用缓存一个 head 的 K 和 V，降低为 MHA 的 KV Cache 大小 1/h。

- **优点:**
  - 节省显存，KV Cache 降低为原始的 1/h，减少计算和通信开销，提升推理速度。
- **缺点:**
  - 性能下降：KV Cache 压缩过于严重，影响模型训练稳定性和模型效果。

04

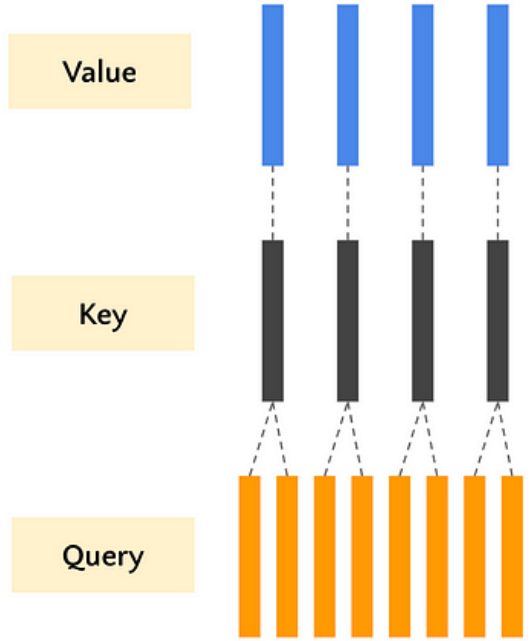
GQA

GQA：Group-Query Attention

- 2023《GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints》。解决 MQA KV Cache 过度压缩，将 KV Head 分为 g 组，每组共享 KV。

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

$$
o_t^{(s)} = \text{Attention}\left(\boldsymbol{q}_t^{(s)}, \boldsymbol{k}_{\leq t}^{([sg/h])}, \boldsymbol{v}_{\leq t}^{([sg/h])}\right) \triangleq \frac{\sum_{i \leq t} \exp\left(\boldsymbol{q}_t^{(s)} \boldsymbol{k}_i^{([sg/h])\top}\right) \boldsymbol{v}_i^{([sg/h])}}{\sum_{i \leq t} \exp\left(\boldsymbol{q}_t^{(s)} \boldsymbol{k}_i^{([sg/h])\top}\right)}
$$



GQA：Group-Query Attention

- 将每一份 K 和 V 均分为 g 组，每组 KV repeat h/g 次，正好实现 h 个 head 所需要 KV。
- 当 g=h 时为 MHA，当 g=1 时为 MQA。llama2/3-70 使用 GQA 设置 g=8，正好每张卡可以负责计算一组 K 和 V 的 Attention, 保证 KV 多样性同时减少卡间通讯。

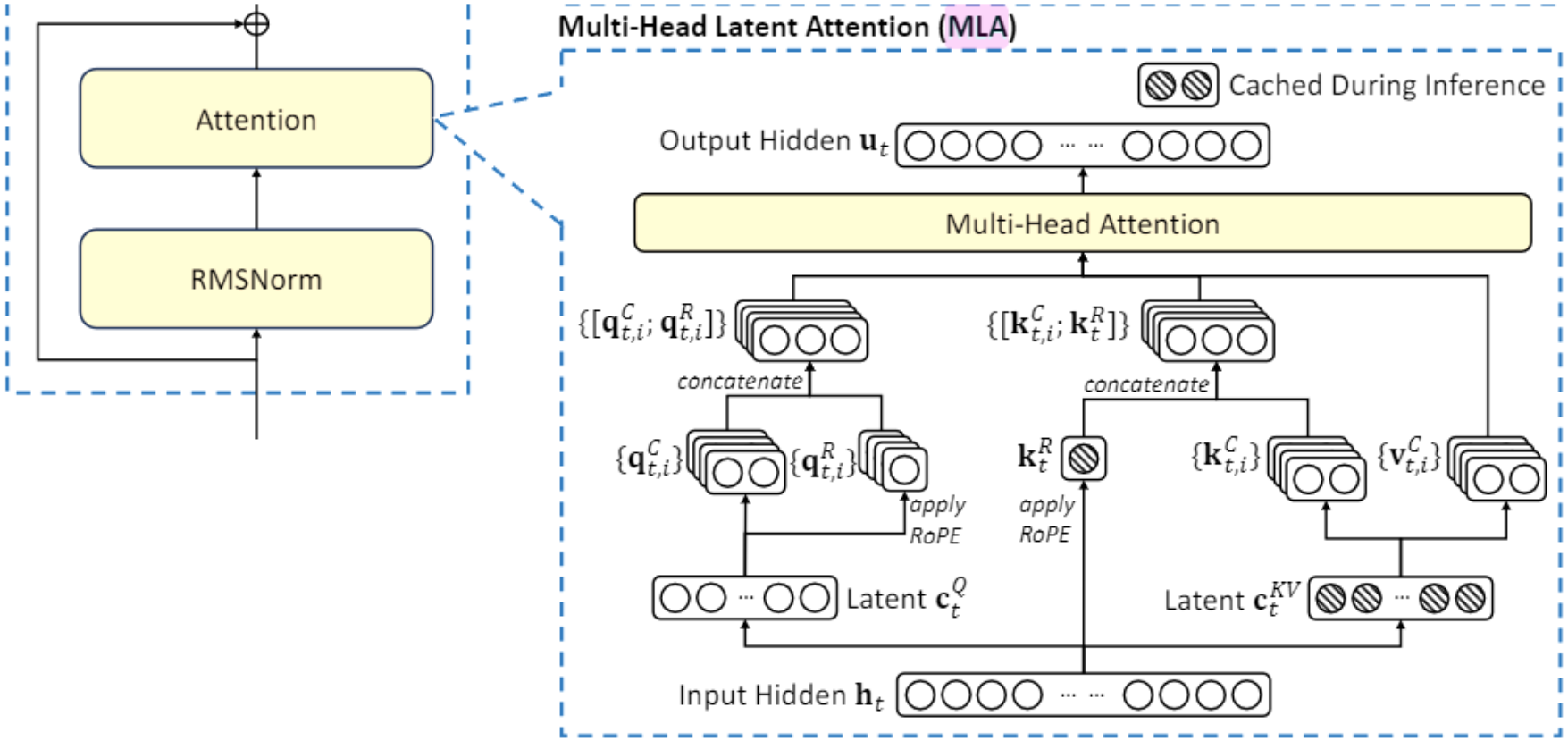
- **优点:**
  - 性能和效率之间平衡：保证 KV 多样性同时，减少 KV Cache 大小；
  - 稳定性：相比 MQA，训练过程较为稳定；
- **缺点:**
  - 需人为合理设置组数 g。

05

MLA

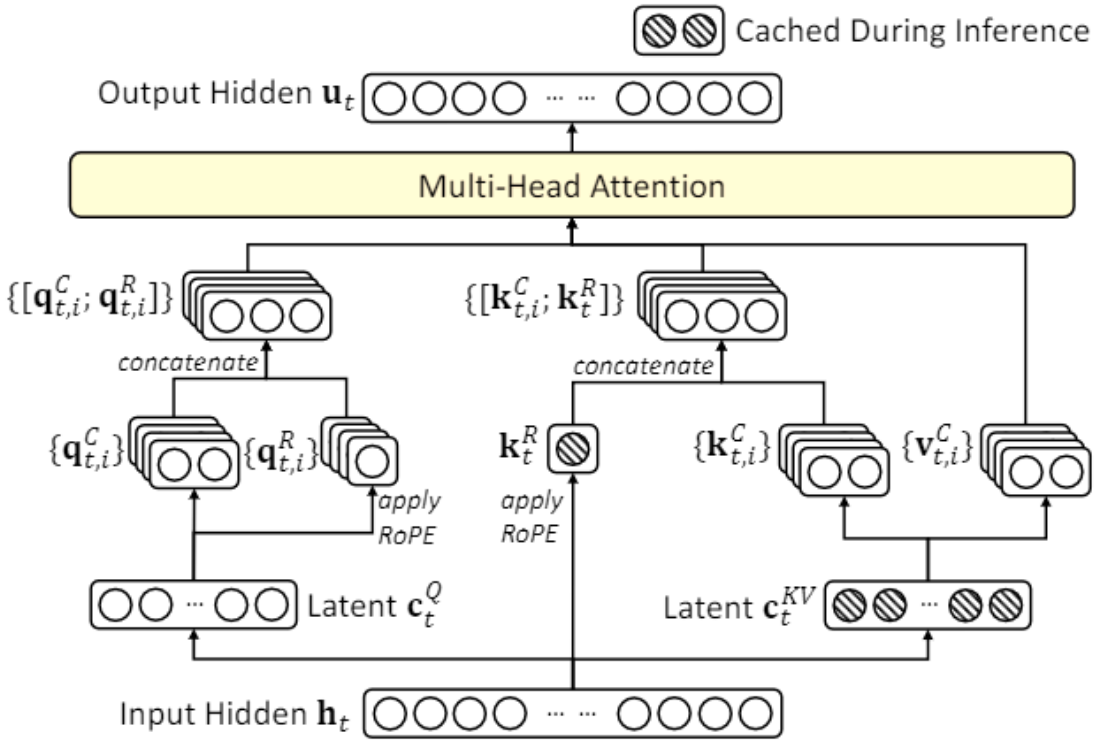
MLA： Multi-head Latent Attention

- MLA 将潜在特征表示纳入注意力机制，以降低计算复杂度并改善上下文表示。核心是对 KV 进行压缩后，再送入标准 MHA 算法中，用更短 KV 向量来进行计算，进而减少 KV Cache 大小。



MLA： Multi-head Latent Attention

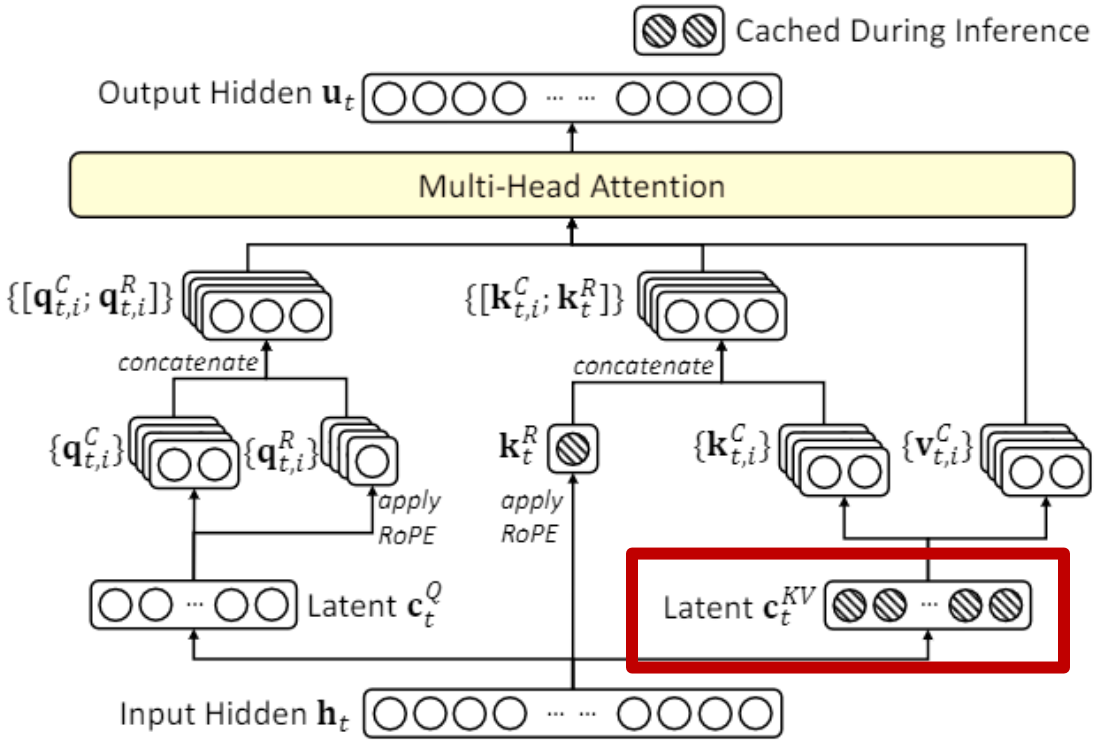
- 通过低秩矩阵将输入 $x_i \in \mathbb{R}^d$ 降维映射为 $c_i \in \mathbb{R}^{d_c}$ ， 于是原始 KV Cache 缓存每个 head 的 $k_i^{(s)}, v_i^{(s)}$ 变为缓存 c_i ；
- 然后通过升维矩阵 和 $\boldsymbol{W}_k^{(s)}$ 和 $\boldsymbol{W}_v^{(s)}$ 将 $c_i \in \mathbb{R}^{d_c}$ 映射为 $k_i^{(s)}, v_i^{(s)}$ ， 后续就执行Attention。



Step1：计算代表 KV Cache 的潜在向量

- c_t^{KV} 是在时间步 t 计算的键值缓存潜在向量。 W^{DKV} 是一个权重矩阵，用于将隐藏状态 ht 映射到键值缓存空间，这一步可以通过神经网络映射得到。c_t^{KV} 相对与原来的 ht 要小很多。

$$\left[c_t^{KV}\right] = W^{DKV}\mathbf{h}_t$$

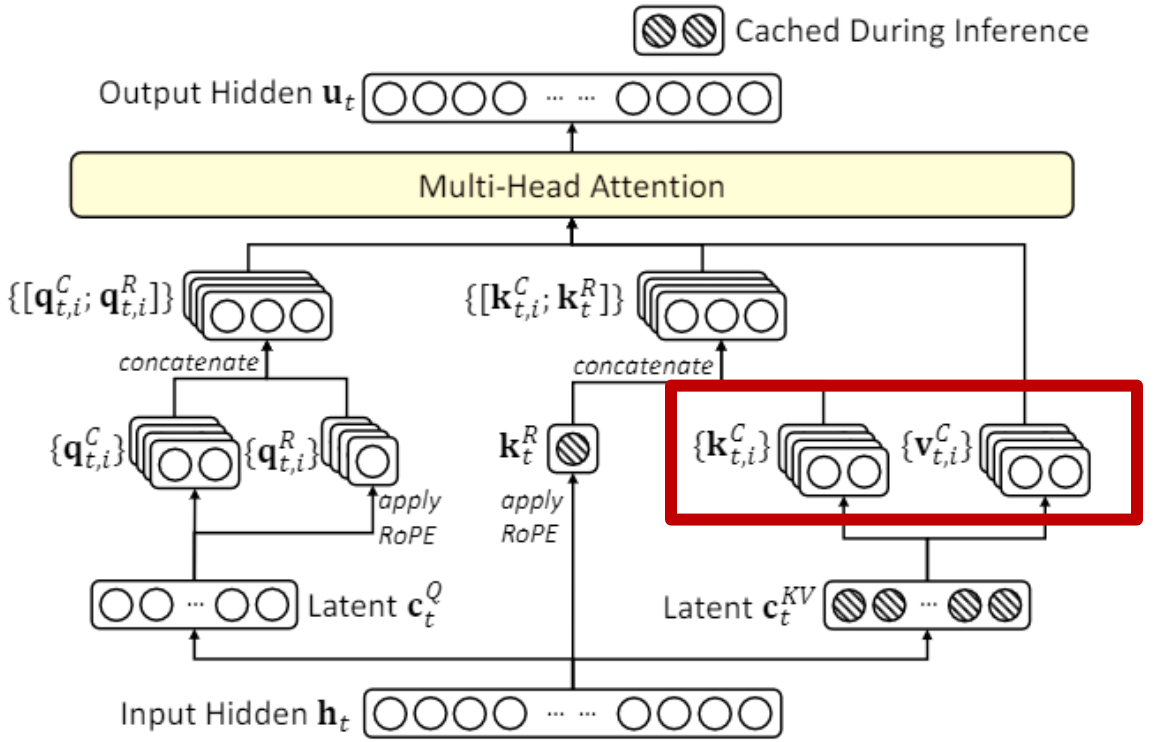


Step2：计算 Key 和 value 潜在向量

- k_t^C 是 Key 潜在向量，通过将 c_t^{KV} 与权重矩阵 W^{UK} 相乘得到，这一步是做上采样，通过潜向量特征 c_t^{KV} 映射得到较大的 k_t^C 用于后续的注意力计算。 v_t^C 计算同理。

$$\left[k_{t,1}^C; k_{t,2}^U; \ldots; k_{t,n_h}^U\right] = k_t^C = W^{UK} c_t^{KV}$$

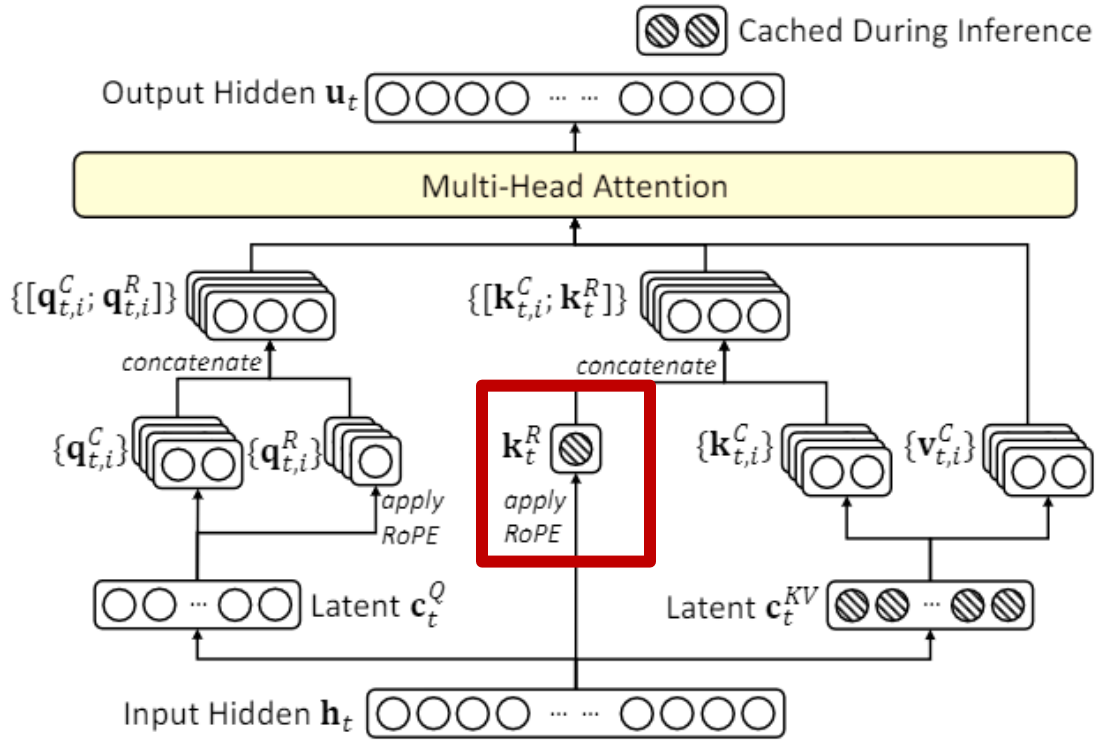
$$\left[v_{t,1}^C; v_{t,2}^U; \ldots; v_{t,n_h}^U\right] = v_t^C = W^{UK} c_t^{KV}$$



Step3：计算旋转位置编码（RoPE）

- RoPE（Rotary Position Embedding）是一种位置编码方法，用于在键向量中引入位置信息

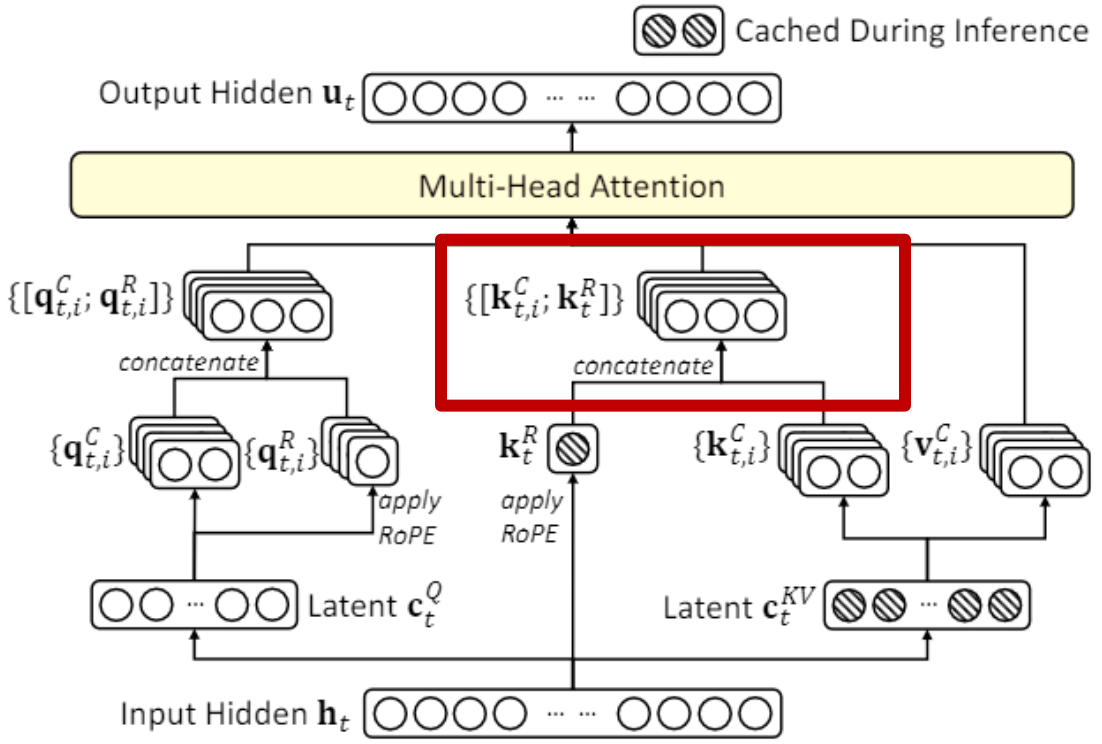
$$k_t^R = \text{RoPE}\left(W^{KR}\mathbf{h}_t\right)$$



Step4：组合潜向量k和位置编码k得到最终的键向量

- 最终的键向量 $k_{t,i}$ 是通过将内容相关的键向量 $k_{t,i}^C$ 和位置编码 k_t^R 连接起来得到。

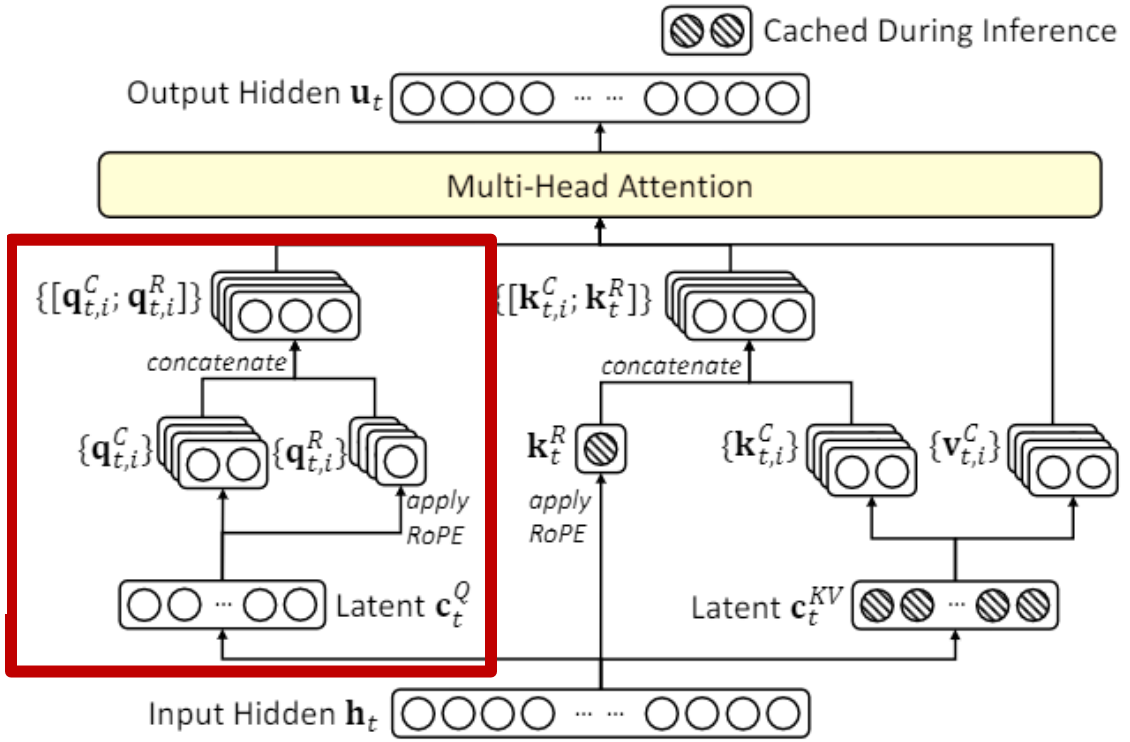
$$k_{t,i} = \left[k_{t,i}^C; k_t^R\right]$$



Step5：计算查询（Query）的潜在向量

- 与 K 向量的计算类似，通过潜在向量计算得到参与后续 MHA 计算的查询向量 q。

$$
\begin{aligned}
\mathbf{c}_t^Q &= W^{DQ}\mathbf{h}_t, \\
[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; \ldots; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C &= W^{UQ}\mathbf{c}_t^Q, \\
[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; \ldots; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R &= \mathrm{RoPE}(W^{QR}\mathbf{c}_t^Q), \\
\mathbf{q}_{t,i} &= [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R],
\end{aligned}
$$

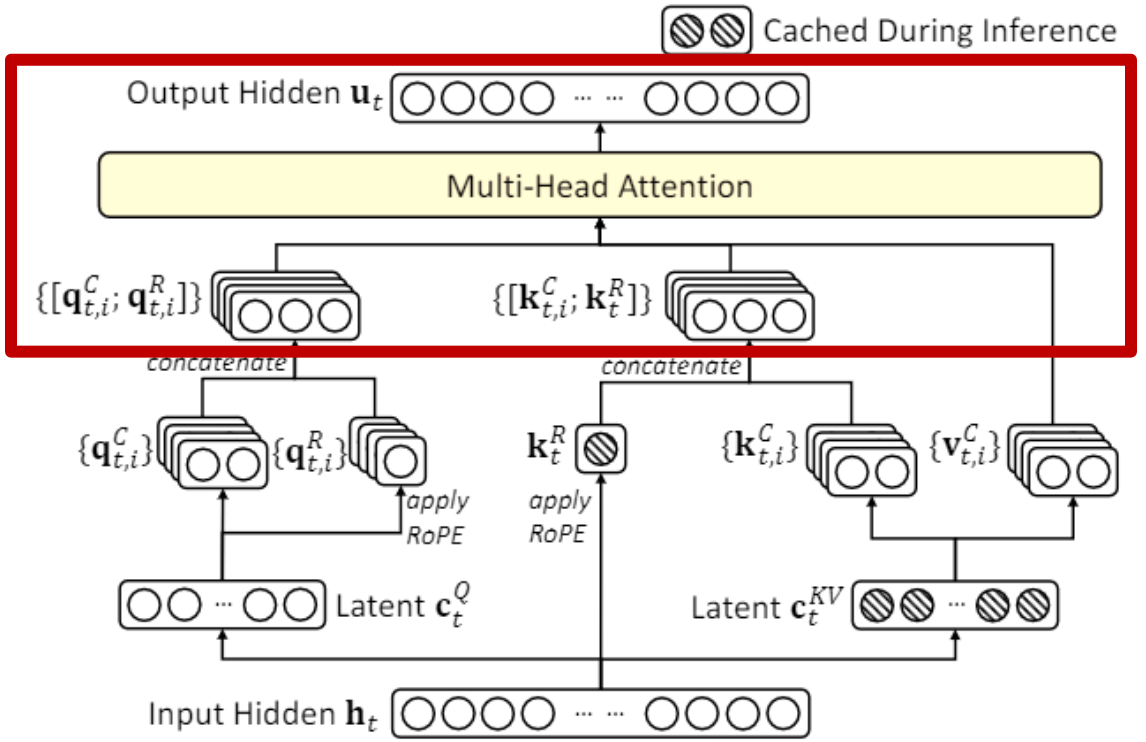


Step6：注意力计算

- 最终的注意力输出 u_t 是通过将查询 $(q_{t,i})$，键 $(k_{t,i})$ 和值 $(v_{j,i}^C)$ 结合起来计算。其中 $o_{t,i}$ 是第 i 个注意力头的输出。

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \text{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}}\right)\mathbf{v}_{j,i}^C,$$

$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \ldots; \mathbf{o}_{t,n_h}],$$



Question?

1. 相较与 MHA，MLA 是再 MHA 之前多了一些 QKV 的处理，为什么会变的更高效?
2. 位置编码信息 RoPE 为什么不直接加在 $k_{t,i}^{C}$ 上，而要新创建一些 k_{t}^{R} ？
3. 为什么对于查询向量 q，也要进行潜向量的计算?

总结与思考

总结

- **Self-Attention 是 Transformer 的核心机制：**
  - 通过计算输入序列中不同位置之间的相关性，实现全局依赖建模；
  - Self-Attention 是 Transformer 模型强大表达能力的关键。

- **为了提升效率与效果，Attention 架构持续演进：**
  - MQA（Multi-Query Attention）：共享多个查询头的键和值，降低计算开销。
  - GQA（Grouped Query Attention）：MQA 与 MHA 的折中方案，兼顾性能与效果。
  - MLA（Multi-head Linear Attention）：探索更高效的注意力计算方式，适应长序列处理。

总结

- **未来趋势：高效、可扩展、适合长上下文**
  - 减少复杂度：随着大模型发展，通过优化 Attention 计算复杂度提出 Linear Attention 等。
  - 长序列建模：结合稀疏注意力与动态路由，进一步压缩KV Cache。
  - 多模态扩展：探索跨模态注意力交互，如视觉-语言联合表征。

Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

Bring AI System to every person, home and organization for a fully connected, intelligent world.

ZOMI

GitHub https://github.com/chenzomi12/AIInfra

引用与参考

- https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers
- https://shangzhih.github.io/jian-shu-attentionji-zhi.html
- https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html
- https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html
- https://www.gnn.club/?p=2729
- http://www.myhz0606.com/article/kv-cache
- https://spaces.ac.cn/archives/10091

- PPT 开源在：https://github.com/chenzomi12/AIInfra