稀疏注意力

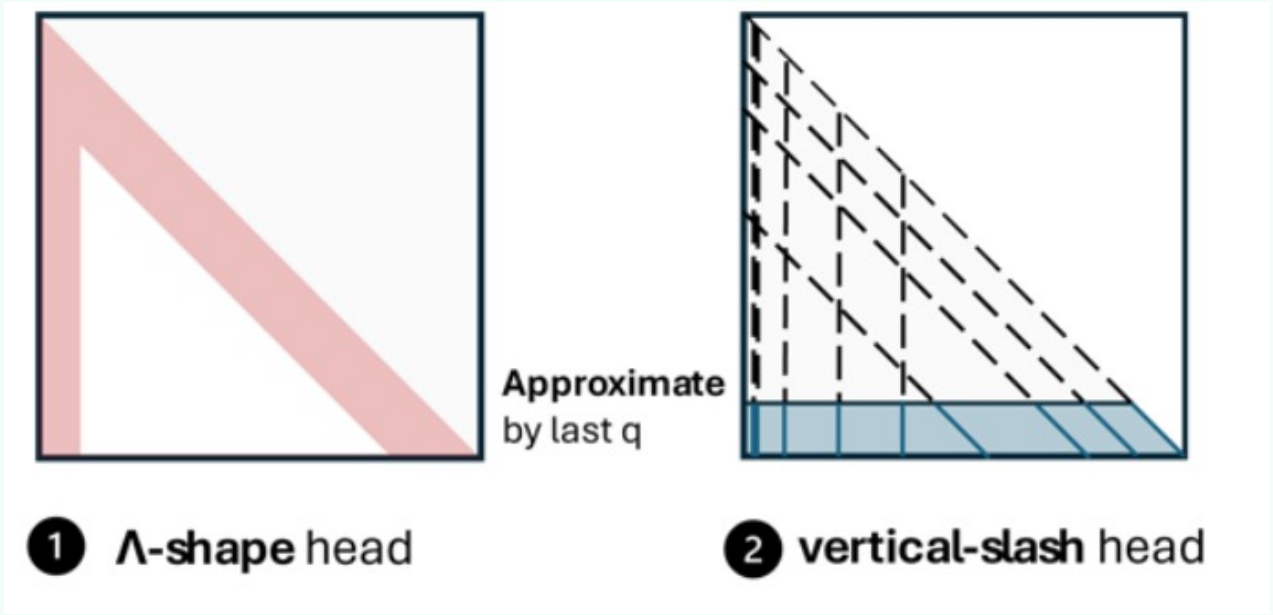
清华大学计算机系 陈键飞

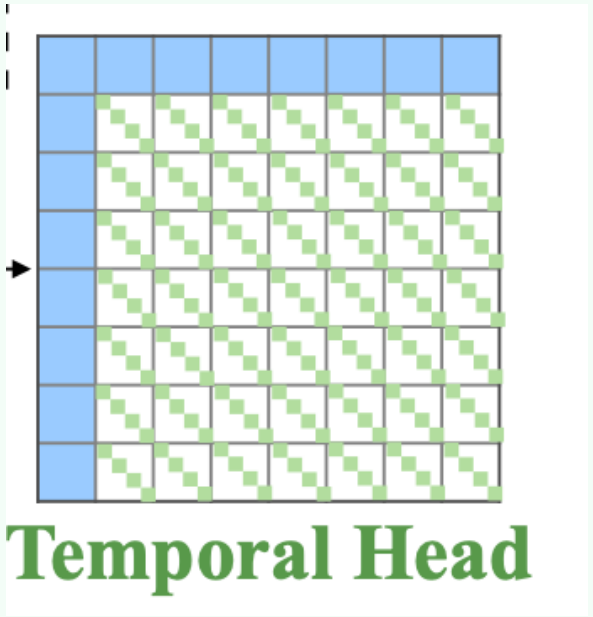
《大模型计算》课程团队

动机

- Attention复杂度终归是$O(N^2)$
- N足够大的时候，终会成为瓶颈
- 理想情况，我们希望有一个无限上下文长度的模型
- 必须要想办法避免N*N的注意力矩阵计算

注意力模式

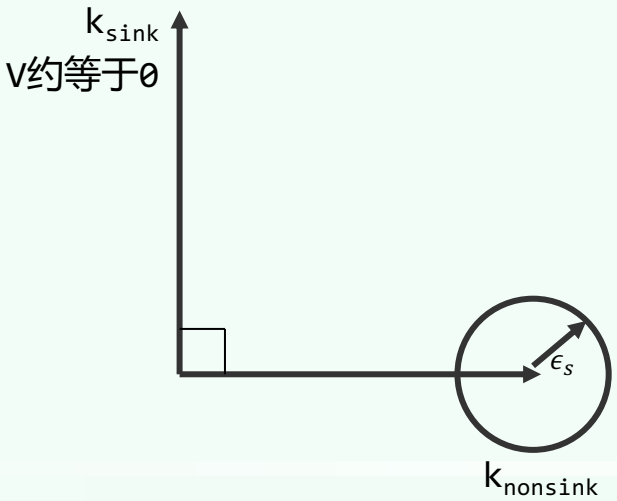




Jiang H, Li Y, Zhang C, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention[J]. arXiv preprint arXiv:2407.02490, 2024.
Xi H, Yang S, Zhao Y, et al. Sparse VideoGen: Accelerating Video Diffusion Transformers with Spatial-Temporal Sparsity[J]. arXiv preprint arXiv:2502.01776, 2025.

Attention Sink

- 注意力权重都在前几个token上，为什么？（定量：cdf > 0.8？）
- 解释：不是每个token的每个head都需要“读取”
- 如果不需要读取，来自attn的O就会稀释residual path上这个token的信息
- 需要一种选择机制，让不同Q token的O magnitude不一样
- 但softmax注意力自带归一化，所有的token的“读取”强度都一样



$$k_{nonsink} = k_0 + \epsilon_s$$
$$q = \alpha k_{sink} + \beta k_0 + \beta \epsilon_s$$
$$\langle q, k_{sink} \rangle = \alpha \|k_{sink}\|^2$$
$$\langle q, k_s \rangle = \beta \|k_0\|^2 + \beta \langle q, \epsilon_s \rangle$$

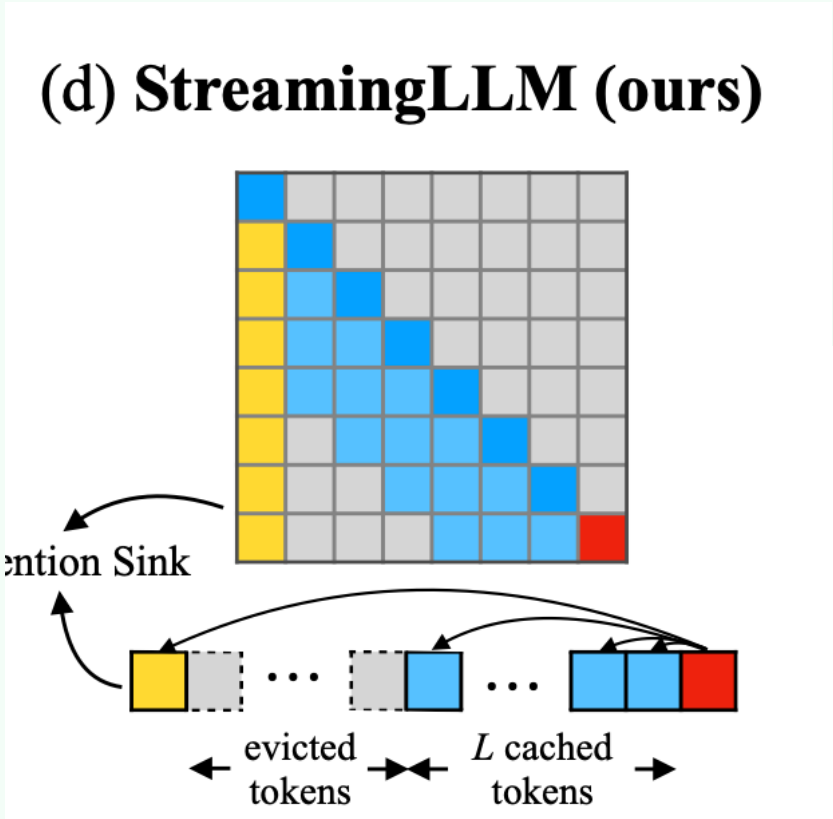
如果想要attention不做事情：让α大β小

如果想让attention做事情：让β大，此时真实起作用的K是ϵ_s

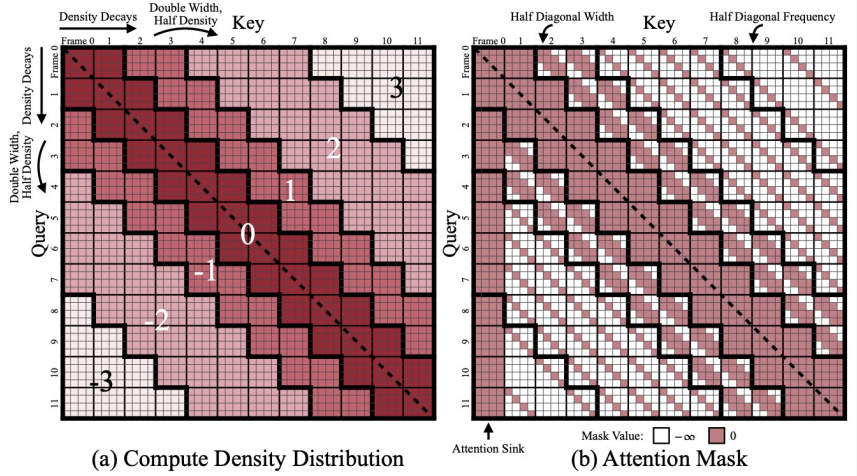
静态稀疏注意力

❖ 分析每个head的attention pattern，给每个head加静态的掩码矩阵

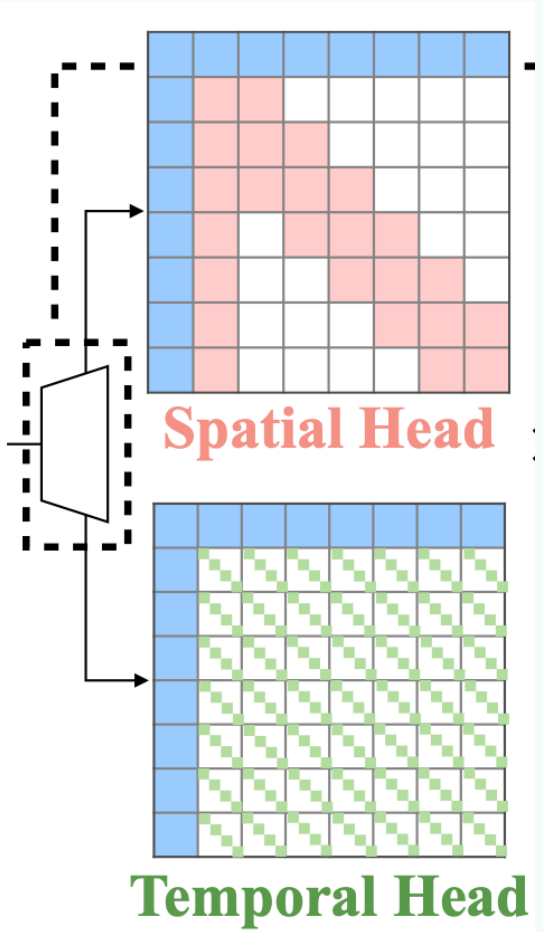
StreamingLLM



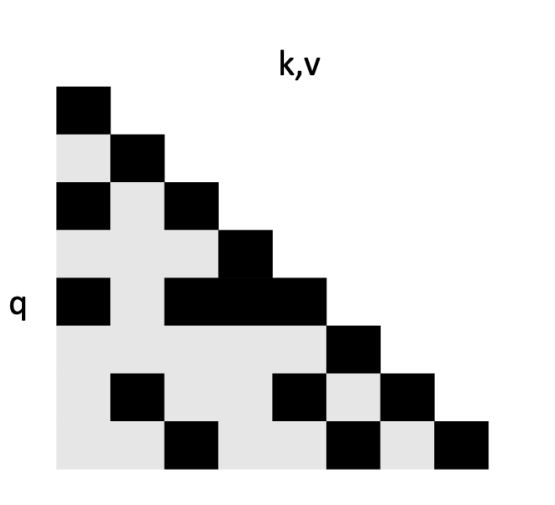
Radial Attention



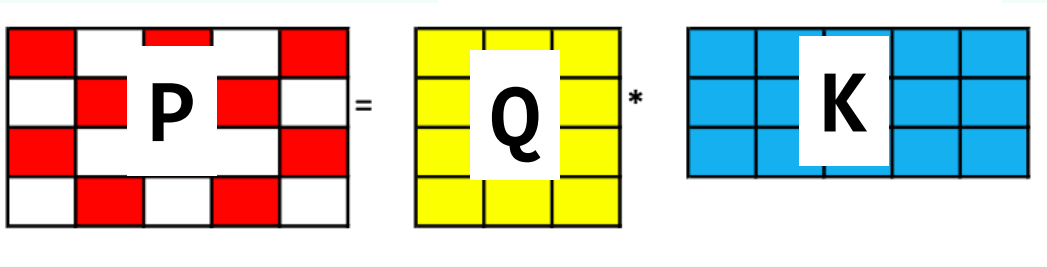
Sparse VideoGen



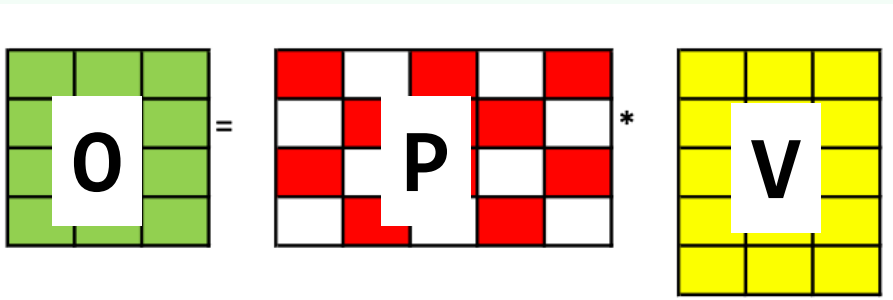
动态稀疏注意力



- 根据数据动态决定注意力掩码
- Idea
  - Identical to SDDMM
  - 只计算选择的注意力分块

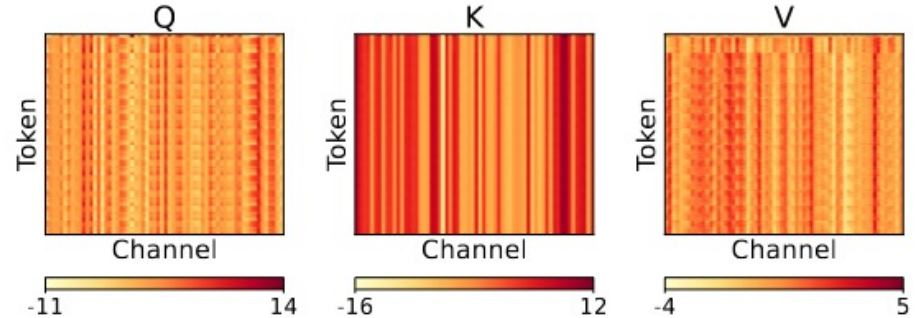


- Questions:
  - How to select the mask/router?
  - How to be hardware friendly?
    - Only computes selected rectangular blocks of the full attention matrix



免训练路由

- Fact: Q/K in adjacent tokens are usually (but not always) similar
- Idea:
  - compress the sequence to a shorter one by mean pooling
  - Compute full attention on the pooled sequence $O(\frac{N^2}{B^2})$
  - Selects topcdf blocks of the pooled attention
  - Compute full attention for the selected blocks $O(\rho N^2)$
- Tricks
  - Do not always compress (tokens might be dissimilar)
  - Don't do topk selection, do topcdf



(a) QKV distribution in Cogvideo

案例：SpargeAttention

SpargeAttn: Accurate Sparse Attention Accelerating Any Model Inference. 2025

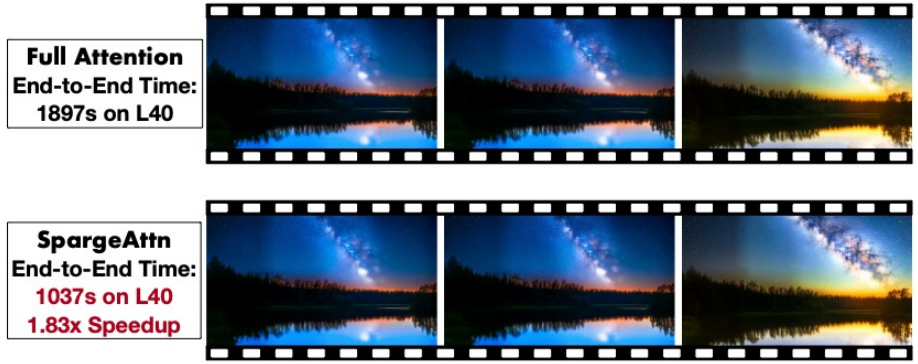


Figure 1. SpargeAttn can achieve 1.83x speedup on Mochi on L40 GPU, with no video quality loss.

- Plug-and-play, lossless inference acceleration
- ~1T flops on RTX 4090
- 6~7x faster than FlashAttention2, even for moderate seq. length 4K~8K
- Supports both LLMs and Diffusion models

Model (seq_len)	**Attention** (Sparsity)	**Speed (TOPS)**↑	**WikiText (Ppl.)** ↓	**Longbench** ↑	**InfiniteBench** ↑	**NIAH** ↑
Llama3.1 (128K)	Full-Attention	156.9	6.013	38.682	0.6594	0.907
	Minference (0.5)	140.1	10.631	28.860	0.5152	0.832
	FlexPrefill (0.5)	240.6	6.476	38.334	0.6460	0.858
	Minference (0.3)	115.7	6.705	34.074	0.6532	0.870
	FlexPrefill (0.42)	206.9	6.067	38.334	0.6581	0.878
	SpargeAttn (0.54)	**708.1**	**6.020**	**39.058**	**0.6638**	**0.909**

案例：MoBA（模型参数可训练，路由不可训练）

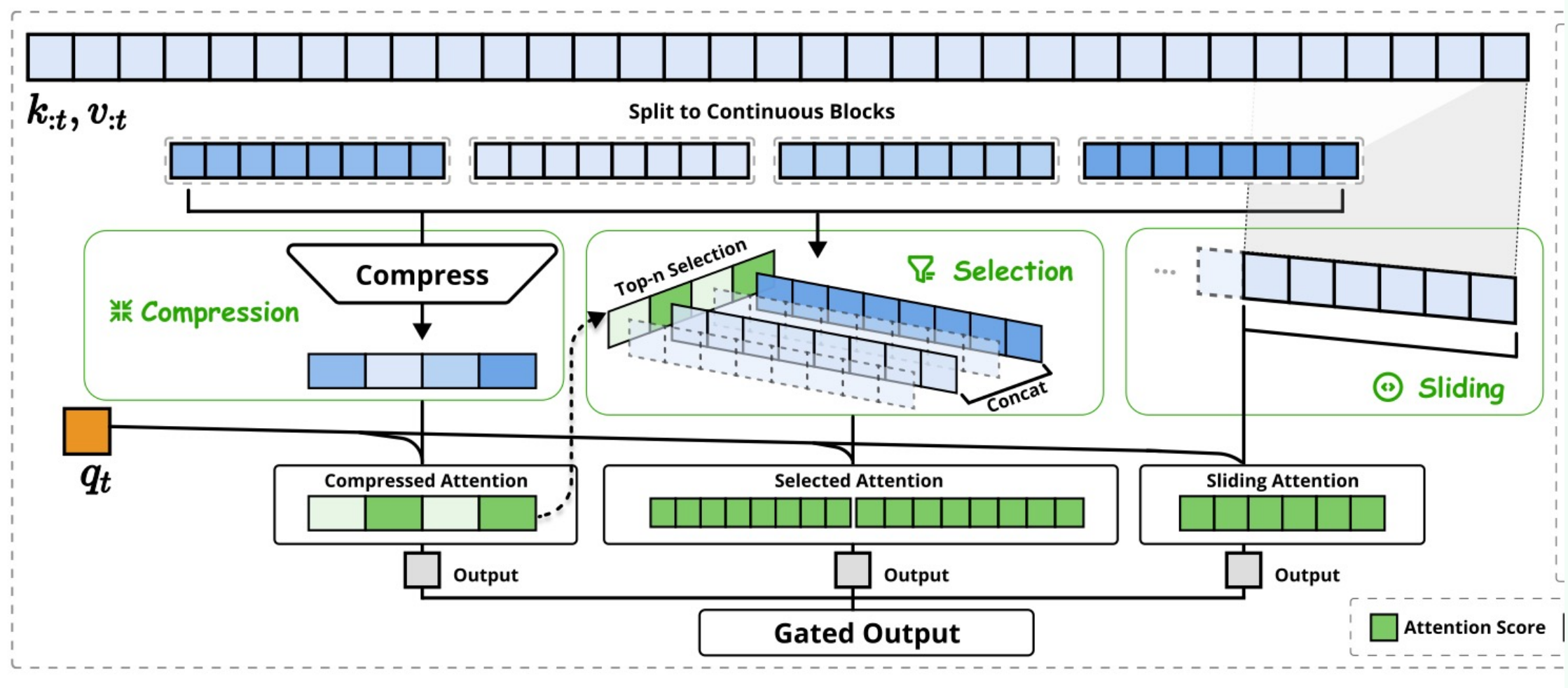
$$\mathrm{MoBA}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Softmax}\left(\boldsymbol{q}\boldsymbol{K}[I]^\top\right)\boldsymbol{V}[I],$$

$$I_i = [(i-1) \times B + 1, i \times B]$$

$$g_i = \begin{cases} 1 & s_i \in \mathrm{Topk}\left(\{s_j | j \in [n]\}, k\right) \\ 0 & \text{otherwise} \end{cases}$$

$$s_i = \langle \boldsymbol{q}, \mathrm{mean\_pool}(\boldsymbol{K}[I_i]) \rangle$$

基于可学习路由的稀疏注意力：NSA



NSA真的需要3个分支吗?

- Sliding window分支按说应该能学出来
- Compression分支很好，但是对于LLM可能更多的是扮演learnable router
  - 对于多模态模型可能有奇效，之后会看到

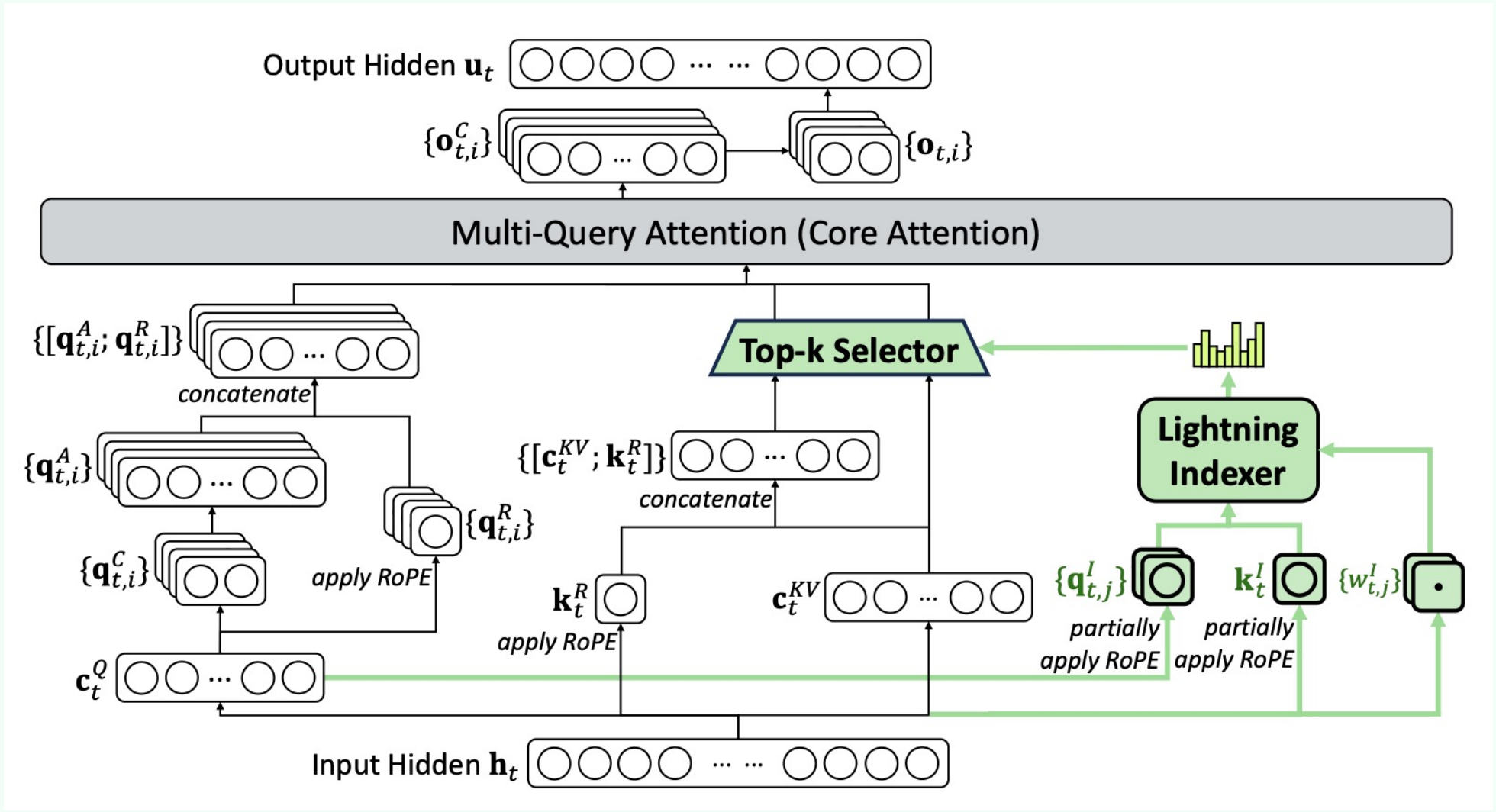
- 此时routing还是以（q1 * k32）为块大小

基于可学习路由的稀疏注意力：DSA

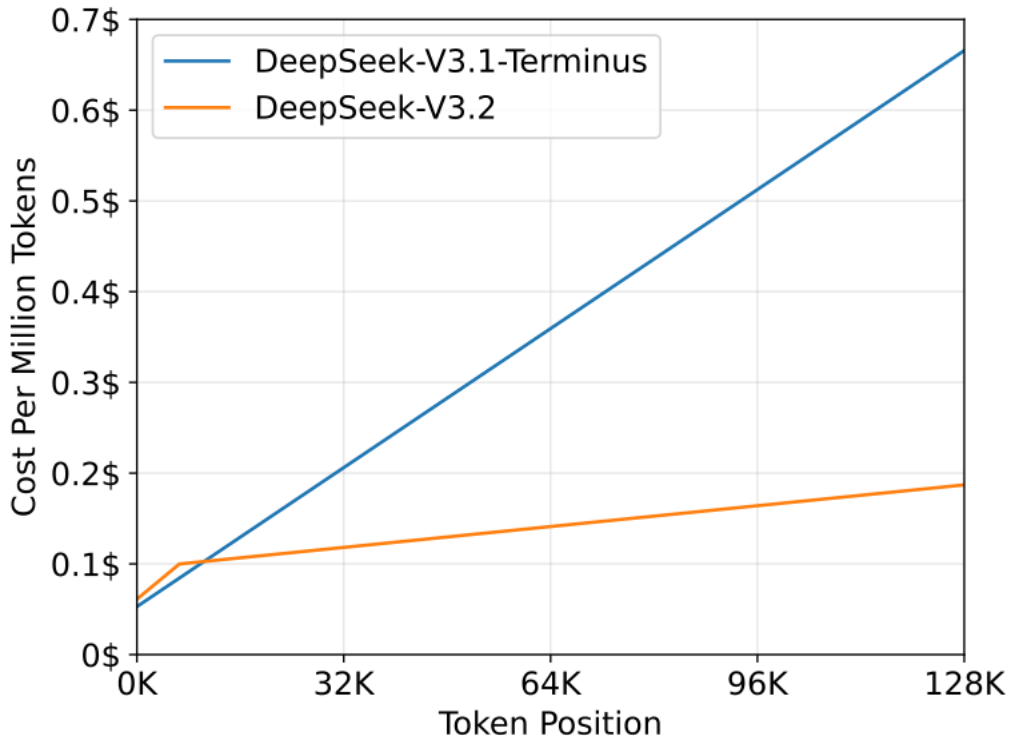
- 直接以（q1 * k1）为block size
- 这么细粒度的block怎么做硬件友好?
- 所有Head一起route，要么都算，要么都不算
- 反正MLA也必须把所有head绑在一起
- 训练不能加速，但反正LLM训练时长文比例不大，索性就不管了
- 用一个小ReLU attention（head少，headdim小）决定哪些k要计算
- 每个Q选top 2048 K
- 设计抉择：按head分组 vs 按token分组

$$I_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \mathrm{ReLU}\left(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I\right),$$

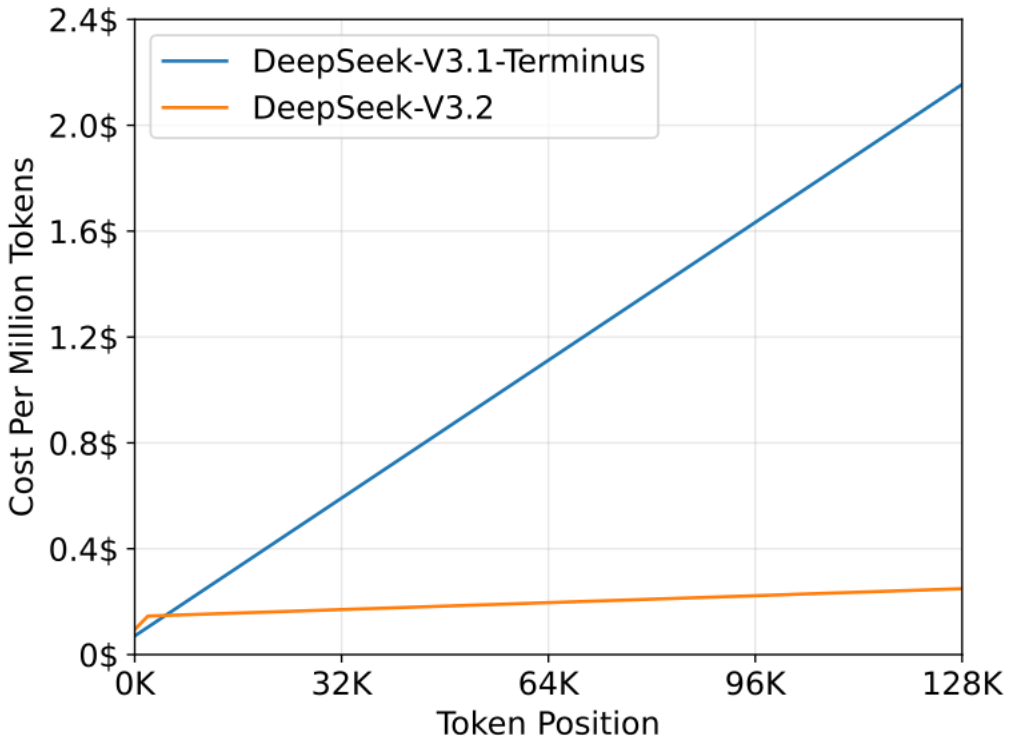
DSA


Output Hidden $\mathbf{u}_t$
$\{\mathbf{o}^C_{t,i}\}$
$\{\mathbf{o}_{t,i}\}$
Multi-Query Attention (Core Attention)
$\{[\mathbf{q}^A_{t,i}; \mathbf{q}^R_{t,i}]\}$
concatenate
$\{\mathbf{q}^A_{t,i}\}$
$\{\mathbf{q}^C_{t,i}\}$
$\{\mathbf{q}^R_{t,i}\}$
apply RoPE
$\mathbf{c}^Q_t$
Top-k Selector
$\{[\mathbf{c}^{KV}_t; \mathbf{k}^R_t]\}$
concatenate
$\mathbf{k}^R_t$
apply RoPE
$\mathbf{c}^{KV}_t$
Lightning Indexer
$\{\mathbf{q}^I_{t,j}\}$
$\mathbf{k}^I_t$
$\{w^I_{t,j}\}$
partially apply RoPE
partially apply RoPE
Input Hidden $\mathbf{h}_t$

DSA



(a) Prefilling



(b) Decoding

Figure 3 | Inference costs of DeepSeek-V3.1-Terminus and DeepSeek-V3.2 on H800 clusters.

DSA有多有效?

- H800: 80GB HBM, 3TB/s, 1000TF@FP16, 2000TF@FP8
- 并行: MoE EP144, Attn DP144, 假设序列全是128K
- 模型显存占用: 前三层FFN+每卡三个Expert+attn=1.2B+7168*2048*3*57*3+11.6B=15.3B=20.3GB
- 每卡最大Batch size=(80-20.3) / (4.2GB/seq) = 14
- 采用bs=14
- MoE延迟（按访存核算） = 读取前三层weight+每层三个expert = 8.7B = 3ms
- MoE延迟（按计算核算） = 总共bs*144个token*61层每层9个expert*7168*2048*3*2 / 144卡总FP8算力 = 0.34ms
- Attn proj（访存）= 读取11.6B FP8参数 / 3TB = 3.87ms
- MLA延迟（假设MLA能打满计算）= (seqlen*heads*576*4) * bs*61 / 1Tflops FP16 = 32.24ms
- 加上DSA后, seqlen固定为2048, **延迟降低为0.52ms**
- 总延迟从3+3.87+32.24=39.11ms降低至3+3.87+0.52=7.39ms, **约5.3x加速**! 解码速度提升至135token/s

线性注意力

清华大学计算机系 陈键飞

《大模型计算》课程团队

动机

- 当$N \to \infty$时，类似DSA的稀疏注意力几乎注定不可能兼顾计算效率和建模能力
- 核心原因：无限长的上下文只能看到有限多个token
- 需要某种“全局表示”，能更紧凑地压缩token中的信息

朴素线性注意力

❖ 平方$O = \mathrm{softmax}(QK^T)V$，改成……

$$O = \underset{N*M}{\phi(Q)}\underset{M*N}{\phi(K)^T}\underset{N*D}{V}$$

❖ ϕ为ReLU。由结合律

$$O = \underset{N*M}{\phi(Q)}(\underset{M*D}{\phi(K)^T V})$$

❖ 时间复杂度由$O(N^2D)$降低至$O(NMD)$，若取D=M=128，则复杂度大幅降低（通常取M=D）

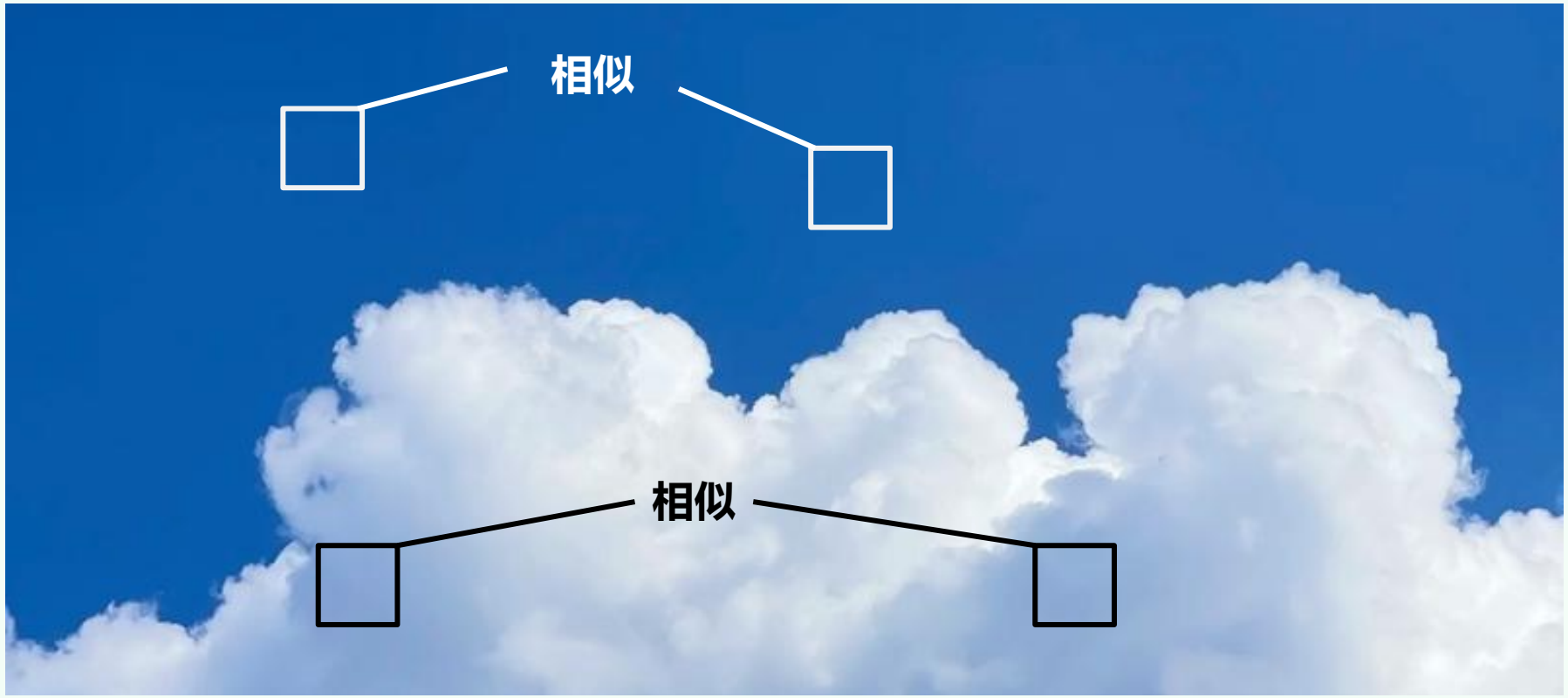
❖ 是什么意思，为什么有效?

线性注意力：解读

- $O = \mathrm{softmax}(QK^T)V$本来是通过注意力矩阵$P$实现N个词元间可学习的、动态的、全连接的混合
- 将一次N*N的混合分解成N->M，M->N的两阶段混合
- 第一阶段：计算$H = \phi(K)^T V$
- 将来自N个词元的信息混合到M个槽位，槽位j读取词元i的强度是动态的、可学习的$\phi(K_{ij})$（非负）
- R是一个“全局记忆体”，所有词元都把信息写进这M个共享槽位
- 第二阶段：计算$O = \phi(Q)H$
- 将来自M个槽位的信息混合到N个词元，词元i读取槽位j的强度是动态的、可学习的$\phi(Q_{ij})$（非负）
- 本质：把来自N个词元的信息**压缩**进M个槽位
- 对比：MLA是将同一个词元的不同head压缩，线性注意力是将不同的词元压缩
- 注意力矩阵$P = \phi(Q)\phi(K^T)$秩限制在M

线性注意力：解读

- 什么时候有效？所有的V： N*K本来就在同一个固定的M维线性子空间（V可压缩）
- 例如说：数据存在大量冗余……



扩展：归一化

归一化

$$o_t = \frac{\phi(q_t)(\phi(K)^T V)}{\text{sum}(\phi(q_t)\phi(K)^T)}$$

分量形式

$$o_t = \frac{\phi(q_t)\sum_{i=1}^{N}\phi(k_i)^T v_i}{\phi(q_t)\sum_{i=1}^{N}\phi(k_i)^T}$$

自回归线性注意力

$$o_t = \frac{\phi(q_t)\sum_{i=1}^{t}\phi(k_i)^T v_i}{\phi(q_t)\sum_{i=1}^{t}\phi(k_i)^T}$$

递归形式

$$\begin{cases} H_t = H_{t-1} + \phi(k_t)^T v_t \\ Z_t = Z_{t-1} + \phi(k_t)^T \\ o_t = \dfrac{\phi(q_t)H_t}{\phi(q_t)Z_t} \end{cases}$$

“状态”：M*D矩阵 + M维向量

自回归注意力：三种计算模式（假设M=D）

❖ 并行模式

$$O = \left((\phi(Q)\phi(K)^T) \circ M\right)V$$

因为自回归掩码M的存在

不再能应用结合率

时间复杂度$O(N^2D)$

❖ 递归模式

$$\begin{cases} H_t = H_{t-1} + \phi(k_t)^T v_t \\ Z_t = Z_{t-1} + \phi(k_t)^T \\ o_t = \dfrac{\phi(q_t)H_t}{\phi(q_t)Z_t} \end{cases}$$

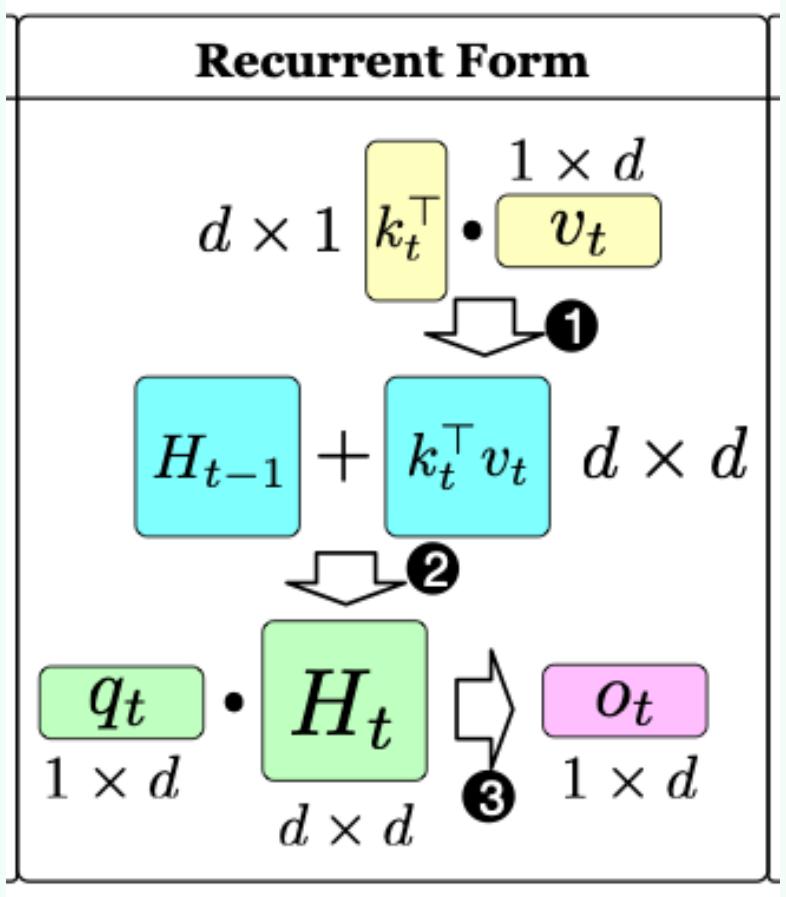
时间复杂度$O(ND^2)$

适用于解码

解码过程中只需维护$O(D^2)$状态

相当于KVcache，但就地更新

没有并行度，不适用于训练



分块模式

- 将序列分成大小为C的块：考虑第c块，对应第1~r个token，则该块中第t个token的注意力可写作

$$o_t = \left(\phi(q_t)\sum_{i=1}^{l}\phi(k_i)^T v_i\right) + \left(\phi(q_t)\sum_{i=l+1}^{t}\phi(k_i)^T v_i\right)$$

$$= \phi(q_t)H_l + \left(\phi(q_t)\sum_{i=l+1}^{t}\phi(k_i)^T v_i\right)$$

- 简洁起见忽略了分母。则第一项可直接利用状态计算，而第二项可并行在$O(CD)$/token时间计算出
- 状态转移

$$H_{r+1} = H_l + \sum_{i=l+1}^{r}\phi(k_i)^T v_i$$

- 总计算时间复杂度：$O(NCD)$（并行计算）+ $O(ND^2)$ （状态转移）
- 并行度为C，增加C可提升并行度，但也会提升复杂度
- 在GPU中可能C=128就够了？其实只有常数代价

遗忘门（DeltaNet）

- 传统写入规则（忽略ϕ）　$H_t = H_{t-1} + k_t^T v_t$
- 问题：记忆只增不减，状态数值越来越大，新知识权重越来越小
- DeltaNet写入规则：覆盖，而非追加

$$H_t = H_{t-1} - \beta_t k_t^T (k_t H_{t-1} - v_t)$$

M*1 1*M M*D 1*D

- $k_t H_{t-1}$：取出待写入位置k_t中的原内容；　β_t：可学习的、输入相关的写入强度
- 当$k_t = a$为独热时，应将v_t写入H的第a行，$k_t H_{t-1}$为取出该行的原内容

- 更一般的情况怎么解释……？

联想记忆

- $H = K^T V = \sum_{i=1}^{N} k_i \otimes v_i$可以叫做一个联想记忆（associative memory）
- 假设$\|k_i\| = 1$
- 该怎么把里面压缩后内容恢复出来?
- $\widetilde{v_i} = k_i H = \sum_{j=1}^{N} \mathrm{cossim}(k_i, k_j)\, v_j$
- 这是一种核回归（kernel regression）
- 如果所有k正交：$\widetilde{v_i} = v_i$，记忆能够精确恢复
- 实际上，因为$N \gg D$，不可能所有N个k两两正交，记忆的v会被与相似的记忆做平滑
- 类比：人会产生既视感（感觉这件事情可能发生过，但实际没有发生过）
- 这种平滑导致我们记忆不精确，但提供了泛化，对模型能力未必完全是坏事

DeltaNet: 解释

- $H_t = H_{t-1} - \beta_t k_t^T (k_t H_{t-1} - v_t)$
- **可以看到：** $v_{\text{old}} = k_t H_{t-1}$**实际上是在读取**$H_{t-1}$**中根据**$k_t$**查询出的记忆**
- **也可写作**
- $H_t = H_{t-1} - \beta_t k_t^T (v_{\text{old}} - v_t)$

测试时训练

- 将H看作一个$k \to v$的函数
- 给定一个k，查到一个v
- 特别地，查询方式为线性探测：$H(k) = kH$

1*M M*D

- 一个好的记忆应满足

$$\min \frac{1}{2} \sum_{i=1}^{N} \|k_i H - v_i\|^2$$

- 给定样本(k_i，v_i)，随机梯度：

$$g_i = (k_i H - v_i)^T k_i$$

- 采用学习率β_t，得到：

$$H_t = H_{t-1} - \beta_t k_t^T (k_t H_{t-1} - v_t)$$

DeltaNet实质上在采用梯度下降法压缩上下文

改进：Gated DeltaNet

- 改写DeltaNet更新

$$H_t = (\mathbf{I} - \beta_t k_t^T k_t) H_{t-1} + \beta_t k_t^T v_t$$

- Gated DeltaNet更新

$$H_t = \alpha_t (\mathbf{I} - \beta_t k_t^T k_t) H_{t-1} + \beta_t k_t^T v_t$$

α_t：遗忘门

- 目标函数

$$H_t = \text{argmin}_H \frac{1}{2} \sum_{i=1}^{N} \|k_i H - v_i\|^2 + \frac{\lambda}{2} \|H\|_F^2$$

- 更新分两步：先对正则化项做梯度下降，再对(k_i，v_i)做SGD
- 遗忘门的本质是“应专注于附近的token”的归纳偏置。实际上就是代替RoPE。GDN不用位置编码
- 但这个归纳偏置会不会太局限了？比如“找到第1000个字符”这样的任务能完成吗？

案例：Mamba2

- 从数学公式来看，实际是朴素线性注意力和GDN的插值
- 带有遗忘门，不带delta rule

$$H_t = \alpha_t H_{t-1} + B_t v_t$$

遗忘强度　　写入强度

- 并行形式可以写作$O = (M \circ QK^T)V$
- 其中M是α_t连乘得到的衰减矩阵，距离越远，衰减越大

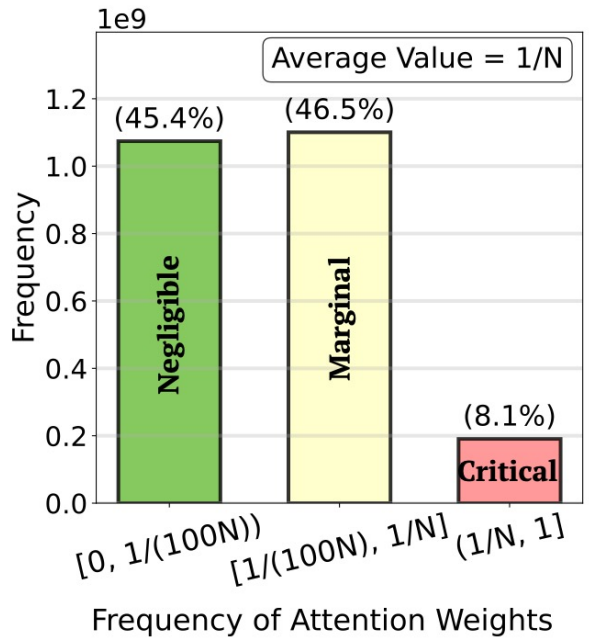
混合高效注意力

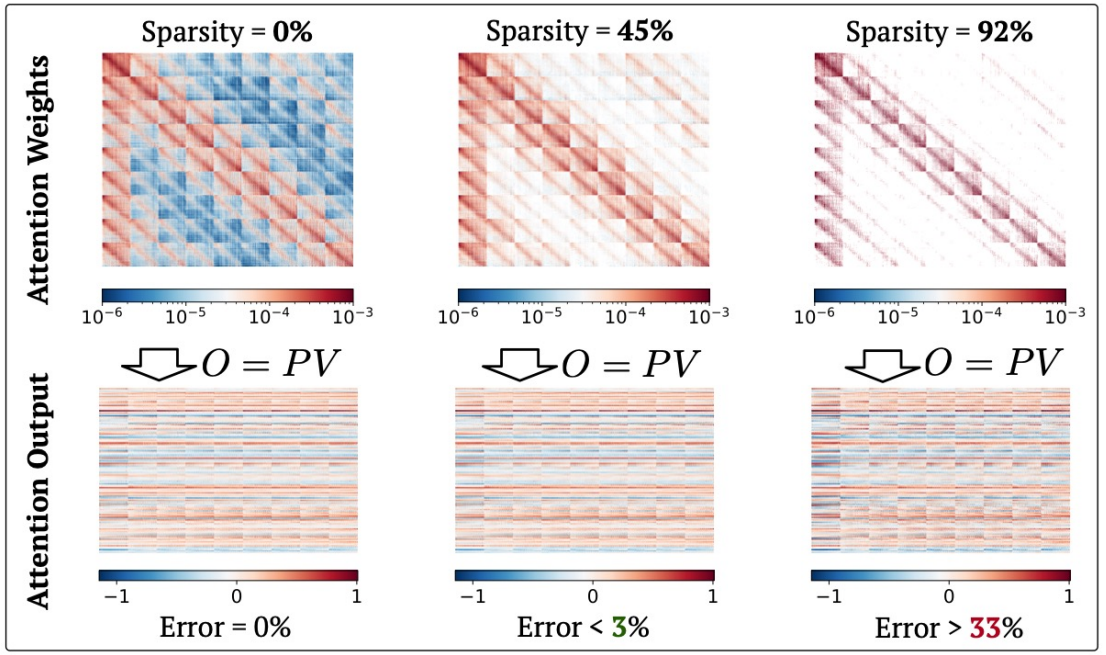
清华大学计算机系 陈键飞

《大模型计算》课程团队

Sparse+Linear Attention

- 实际上，attention map不见得是sparse的……
- 如果序列长度增加，我们希望sparsity能提升
- 但其实提升不了

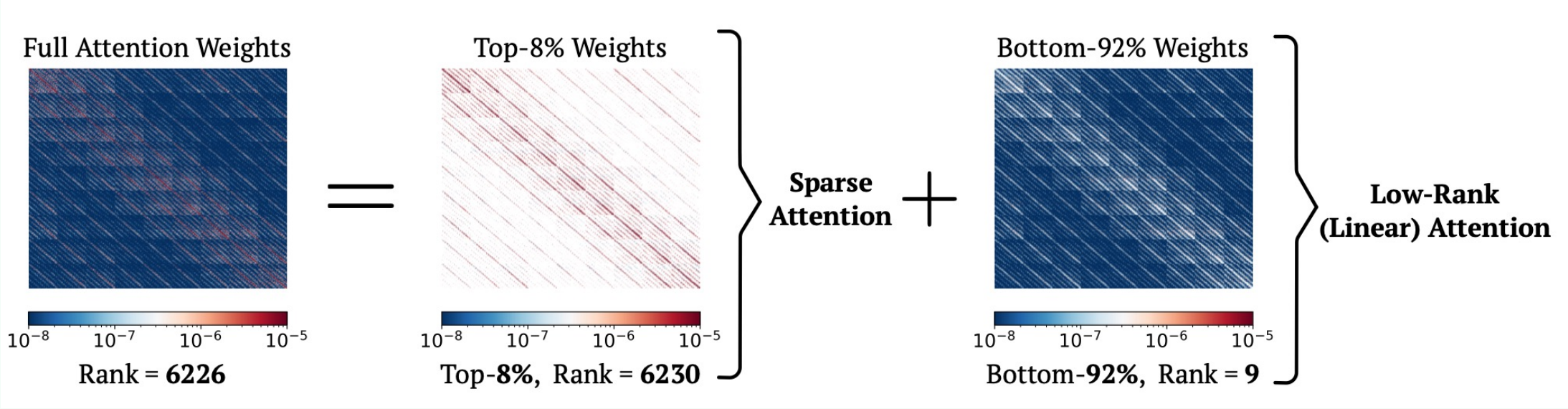




Sparse+Linear Attention

但可能有很多token是相似的

因此可以用linear分支来吸收不稀疏的部分



Sparse+Linear Attention

❖ SLA

$$\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^\top / \sqrt{d}, \quad \mathbf{P}_{ij} = \mathrm{OnlineSoftmax}(\mathbf{S}_{ij}), \quad \mathbf{O}_i^s = \mathbf{O}_i^s + \mathbf{P}_{ij} \mathbf{V}_j.$$

$$\frac{\phi(Q)\phi(K)^\top}{\mathrm{rowsum}(\phi(Q)\phi(K)^\top)} \odot (1 - M).$$

$$O = O^s + \mathrm{Proj}(O^l).$$

结果

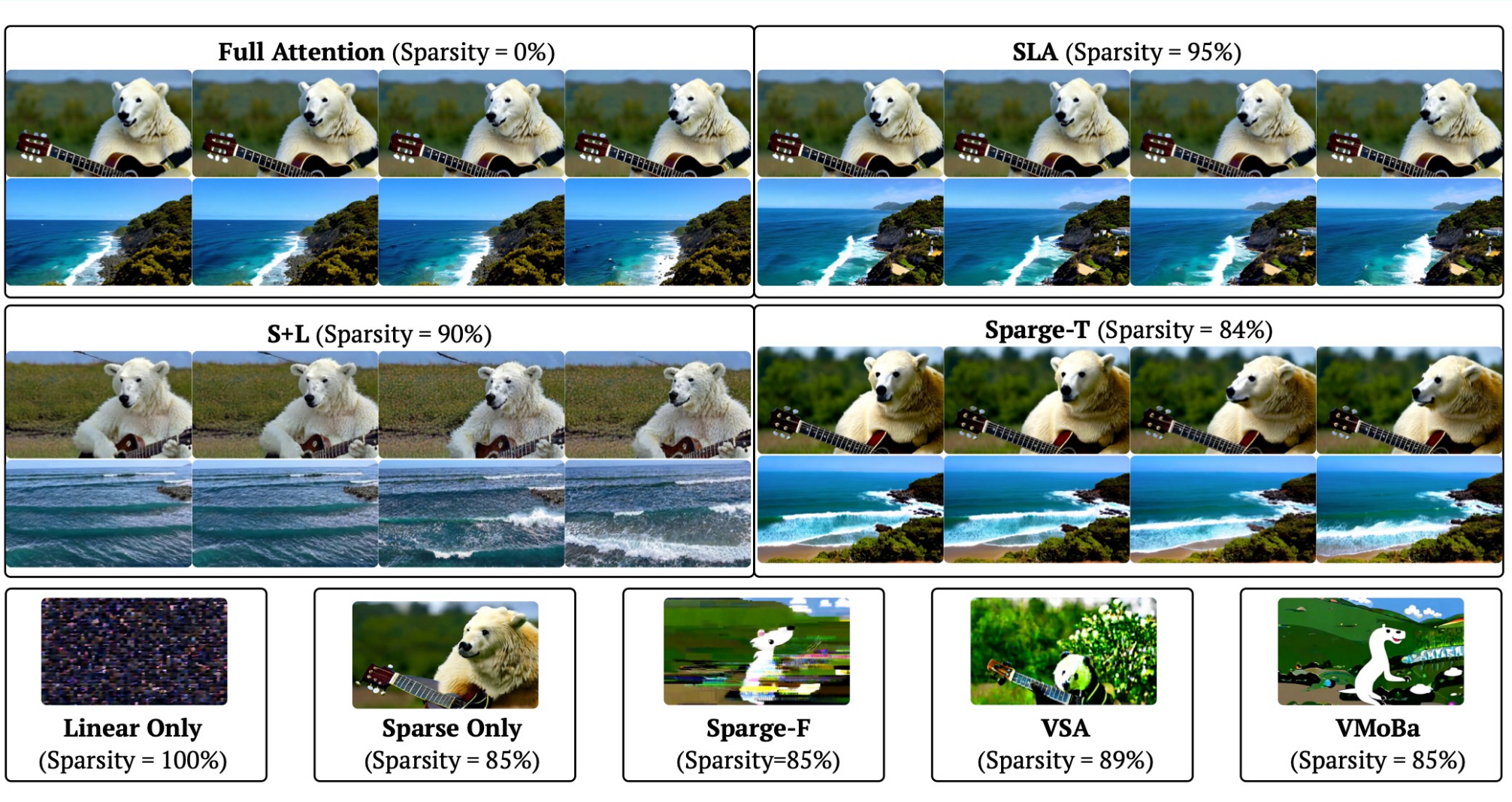
Wan2.1-1.4B可以做到95%的sparsity

Table 1: Quality and efficiency comparison of SLA and other baseline methods.

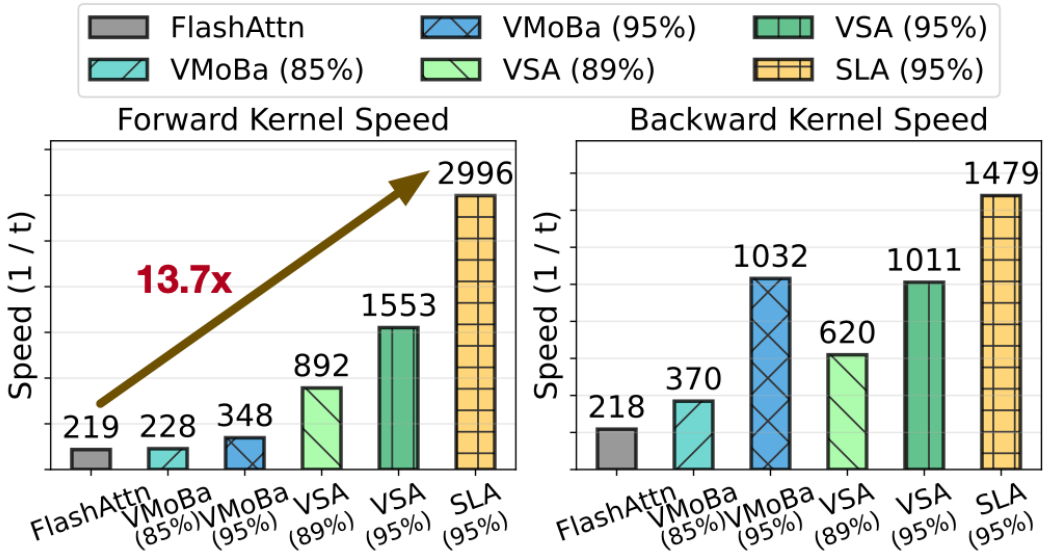
Method	Quality							Efficiency	
	VA ↑	VT ↑	IQ ↑	OC ↑	AQ ↑	SC ↑	VR ↑	FLOPs ↓	Sparsity ↑
Full Attention	76.78	82.88	62.5	23.3	56.1	93.0	0.059	52.75T	0%
Sparge-F	0.002	0.026	26.0	4.6	35.7	85.1	-0.216	7.91T	85%
Sparge-T	73.83	77.87	61.9	22.7	55.4	93.1	0.014	7.38T	84%
VMoBa	32.33	35.79	58.0	18.8	46.2	89.9	-0.175	7.91T	85%
VSA	55.37	64.61	60.6	22.4	51.9	83.6	-0.069	5.92T	89%
SLA	**76.96**	**83.92**	**62.2**	**23.6**	**55.9**	93.1	**0.048**	**2.74T**	**95%**

结果

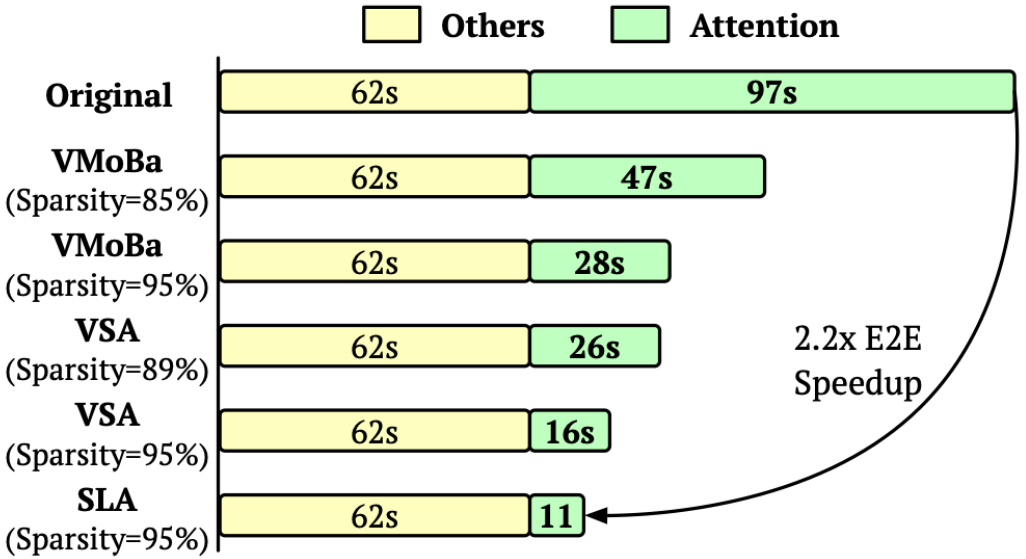
而只用sparse不用linear完全不行



结果



(a) Attention GPU Kernel speed comparison on RTX5090.



(b) End-to-end video generation latency comparison.

延伸阅读

- https://attention-survey.github.io/files/Attention_Survey.pdf

谢谢

清华大学计算机系 陈键飞

《大模型计算》课程团队