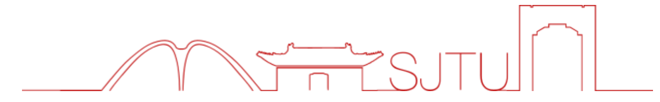


大语言模型的高效计算

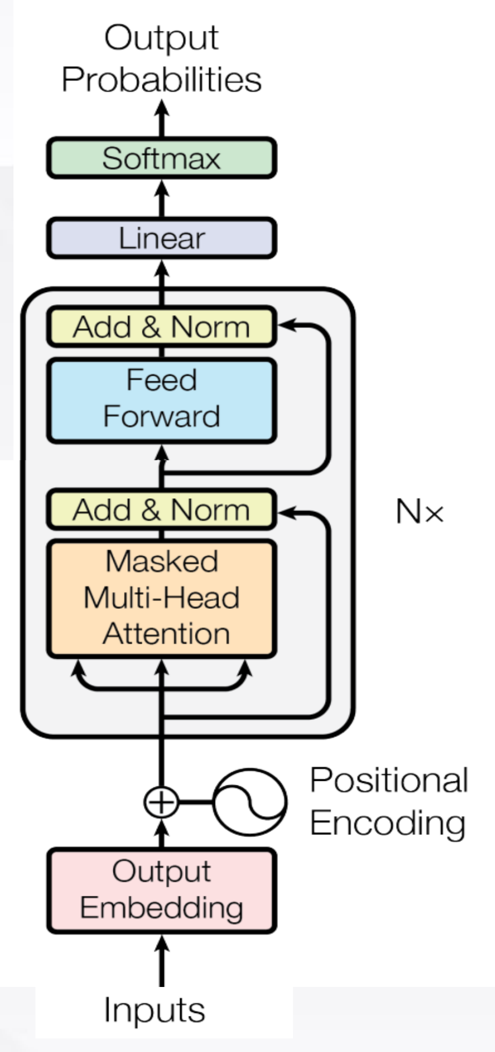
林洲汉

上海交通大学 LUMIA实验室

2024年12月1日

饮 水 思 源 • 爱 国 荣 校

目录



1 • Attention机制层面

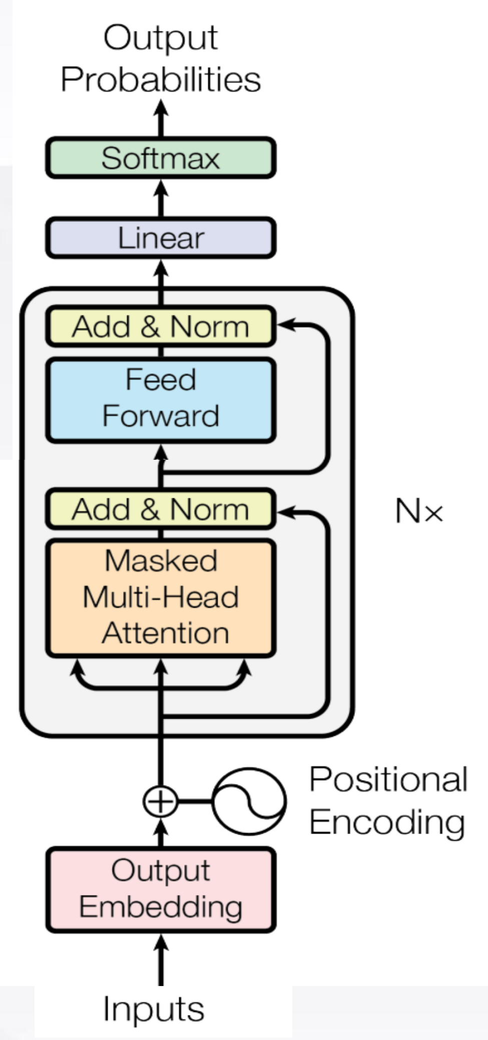
2 • 逐层KV Cache压缩层面

3 • 层间KV Cache压缩层面

4 • 宏观计算架构适配层面

SJTU

目录



1

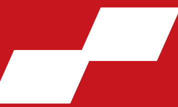
- Attention机制层面

这一层面的方法对模型的加速，集中在对attention机制本身所涉及的各个要素（如Q、K、V、softmax计算方式等）的改进。

- Multi-Query Attention
- Group-Query Attention
- Multi-head Latent Attention
- Palu
- Performers

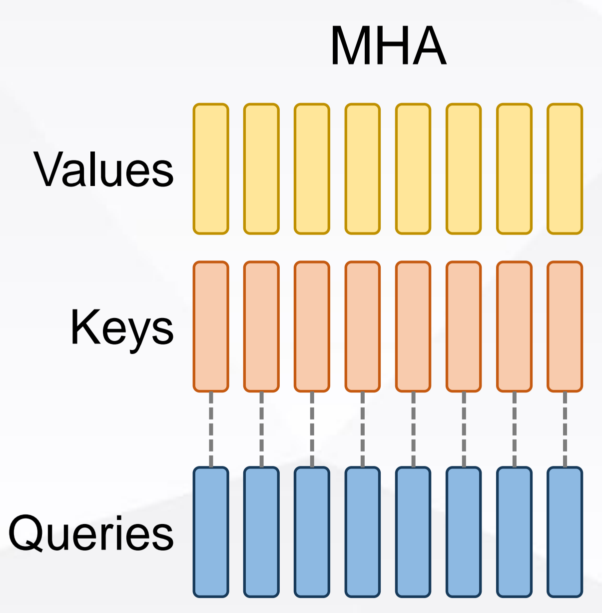
SJTU

回顾：原始的Multi-Head Attention (MHA)



Multi-Head Attention (MHA)

$$o_j^{(s)} = Attention(q_j^{(s)}, k_{\leq j}^{(s)}, v_{\leq j}^{(s)}) = softmax\left(\frac{q_j^{(s)} k_{\leq j}^{(s)T}}{\sqrt{d_h}}\right) v_{\leq j}^{(s)}$$



- 每个token拥有一组Query, Key, Value
- KV cache per token：$2 \times n_h \times d_h \times l$

Head数量 (n_h)，Head维度 (d_h)，层数 (l)

加速策略：

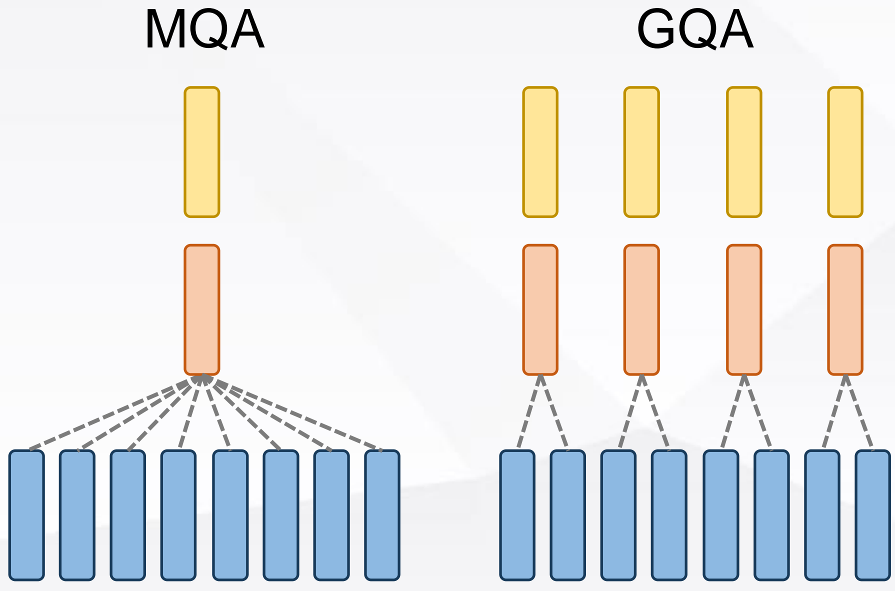
- 减少 head数量n_h
- 降低 head维度d_h

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need." Advances in Neural Information Processing Systems, 2017.

两种加速策略概览

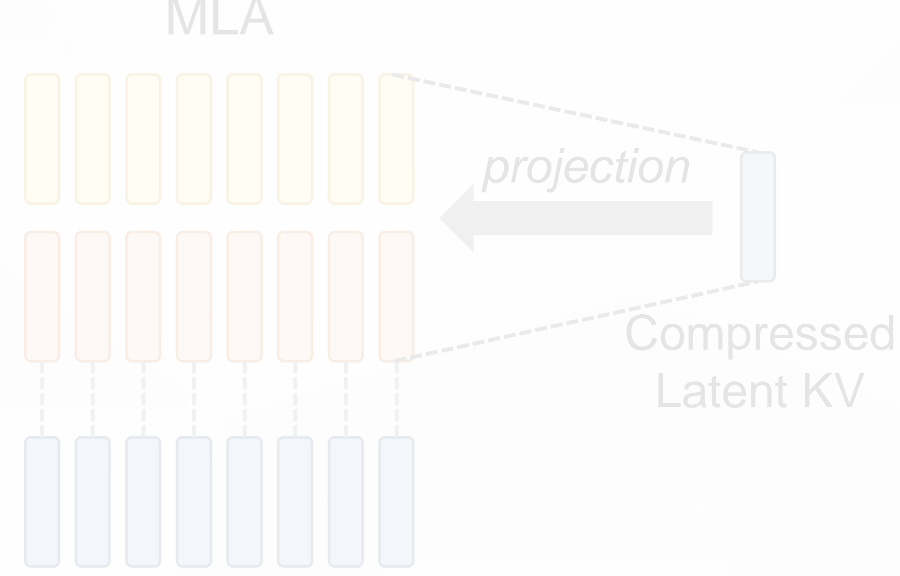
减少
head数量n_h

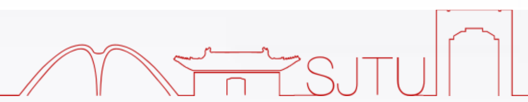
- 不同的query head共享同一套KV
- MQA, GQA



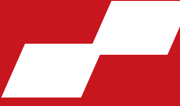
降低
head维度d_h

- 存储低秩压缩后的KV
- MLA

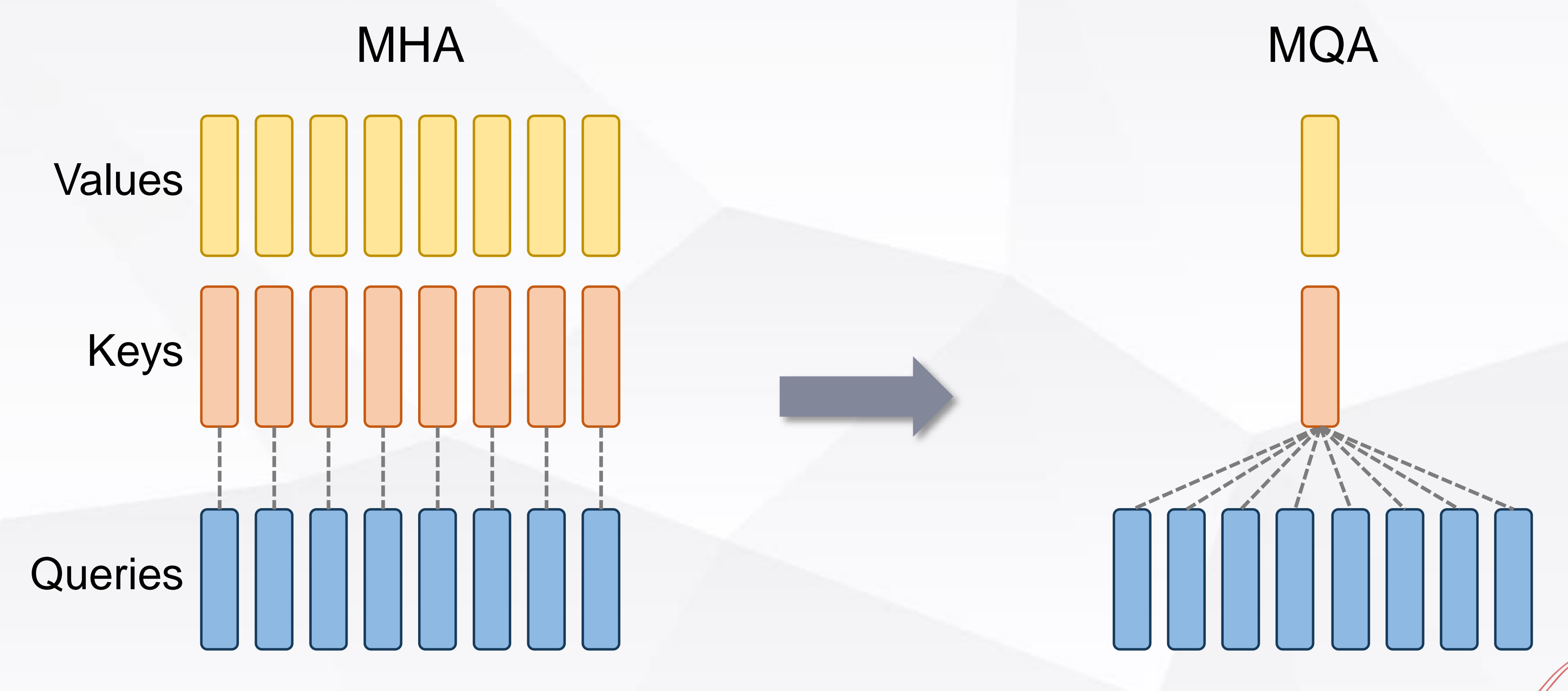




Multi-Query Attention (MQA)



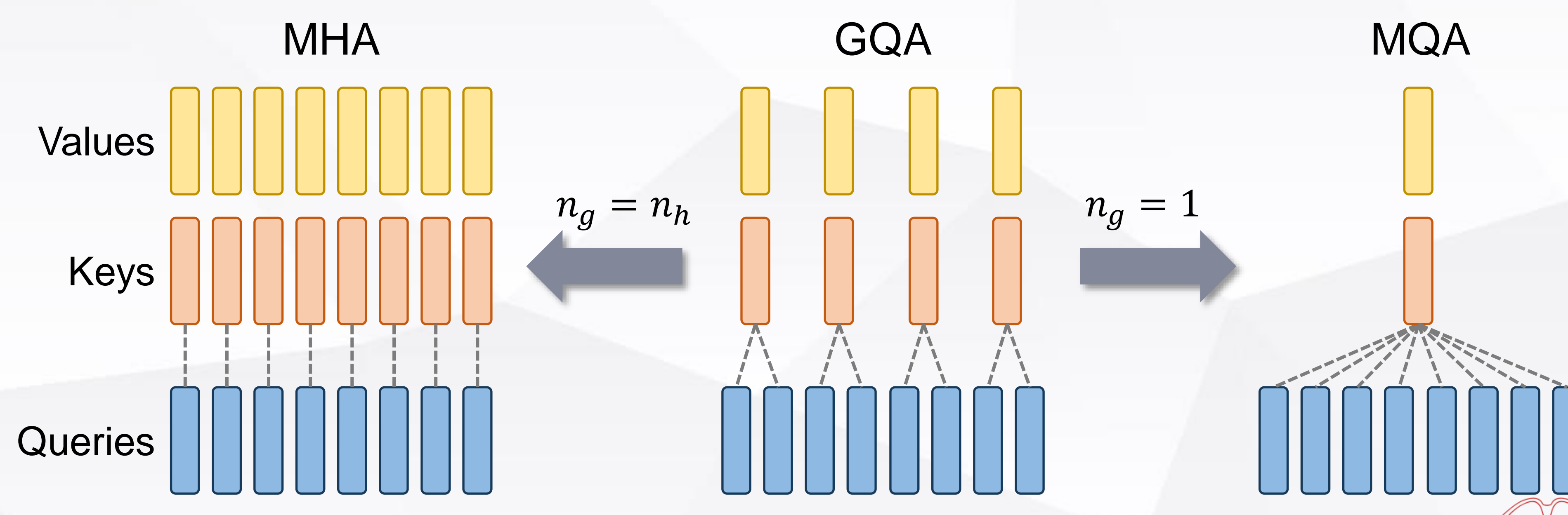
- 所有attention head共享同一组KV
- KV cache per token：$2d_h l$
- 应用在PaLM (2022), StarCoder (2023), Gemini (2024)等模型上



Shazeer N. Fast transformer decoding: One write-head is all you need[J]. arXiv preprint arXiv:1911.02150, 2019.

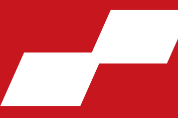
Grouped-Query Attention (GQA)

- 将n_h个head分成n_g组，组内共享同KV
- KV cache per token：$\boldsymbol{2n_g d_h l}$
- 当$n_g = n_h$时，GQA → MHA；当$n_g = 1$时，GQA → MQA
- 应用在LLaMA2-34B, 70B (2023), LLaMA3 (2024), DeepSeek-v1 (2024)等模型上



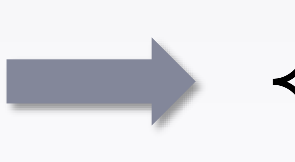
Ainslie J, Lee-Thorp J, de Jong M, et al. Gqa: Training generalized multi-query transformer models from multi-head checkpoints[J]. arXiv preprint arXiv:2305.13245, 2023.

Grouped-Query Attention (GQA)



- Meta在LLaMA2-30B上的实验，GQA在多个任务上取得了与MHA接近的表现

这里在比较时保持了总参数量与MHA接近

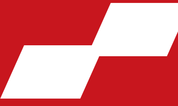


- MQA的FFN维度扩大到1.33倍
- GQA的FFN维度扩大到1.3倍
- 增加MQA和GQA模型的层数

	BoolQ	PIQA	SIQA	Hella-Swag	ARC-e	ARC-c	NQ	TQA	MMLU	GSM8K	Human-Eval
MHA	**71.0**	**79.3**	48.2	75.1	71.2	**43.0**	12.4	44.7	**28.0**	4.9	**7.9**
MQA	70.6	79.0	47.9	74.5	71.6	41.9	**14.5**	42.8	26.5	4.8	7.3
GQA	69.4	78.8	**48.6**	**75.4**	**72.1**	42.5	14.0	**46.2**	26.9	**5.3**	**7.9**

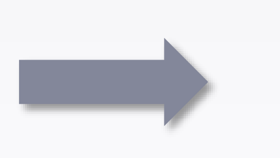
Touvron H, Martin L, Stone K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint arXiv:2307.09288, 2023.

Grouped-Query Attention (GQA)



- 但是，DeepSeek在7B dense模型上的实验，MHA的表现显著优于MQA与GQA

这里在比较时保持了
总参数量与MHA接近



MQA的FFN维度扩大到1.33倍
GQA的FFN维度扩大到1.3倍
增加MQA和GQA模型的层数

Benchmark (Metric)	# Shots	Dense 7B w/ MQA	Dense 7B w/ GQA (8 Groups)	Dense 7B w/ MHA
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	**37.0**
MMLU (Acc.)	5-shot	37.9	41.2	**45.2**
C-Eval (Acc.)	5-shot	30.0	37.7	**42.9**
CMMLU (Acc.)	5-shot	34.6	38.4	**43.5**

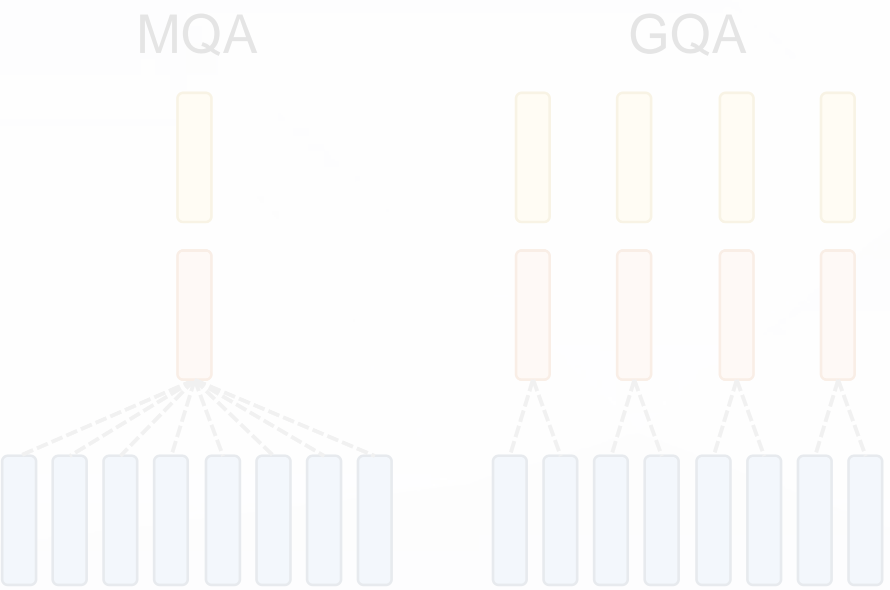
Liu A, Feng B, Wang B, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model[J]. arXiv preprint arXiv:2405.04434, 2024.

SJTU

两种加速策略概览

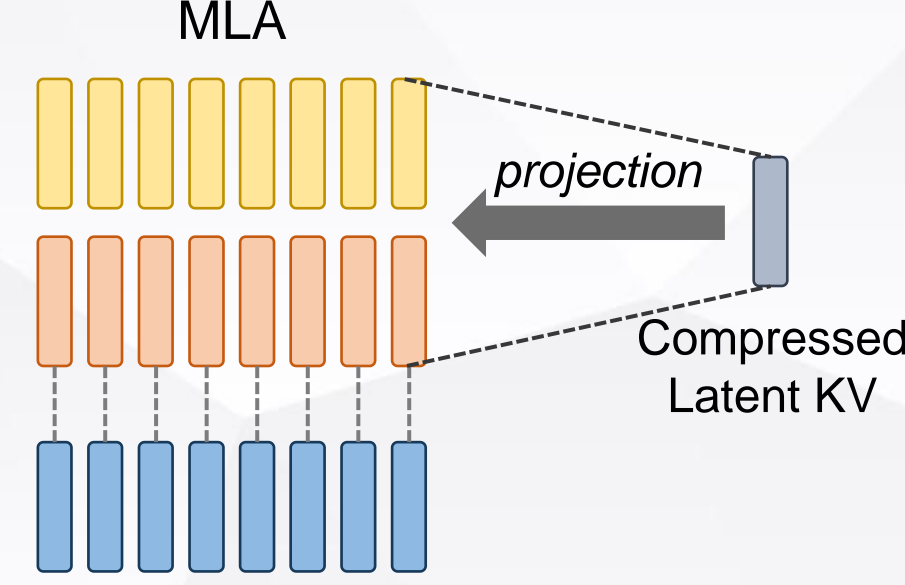
减少
head数量n_h

- 不同的query head共享同一套KV
- MQA, GQA



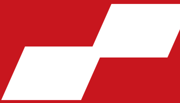
降低
head维度d_h

- 存储低秩压缩后的KV
- MLA



SJTU

Multi-head Latent Attention (MLA)



- Low-rank joint compression for KV Cache:

$W_Q, W_K \in \mathbf{R}^{d \times n_h d_h}$

$h_i \in \mathbf{R}^d$ × Projection Matrices $W_Q(W_K)$ = $k_i, v_i \in \mathbf{R}^{n_h d_h}$

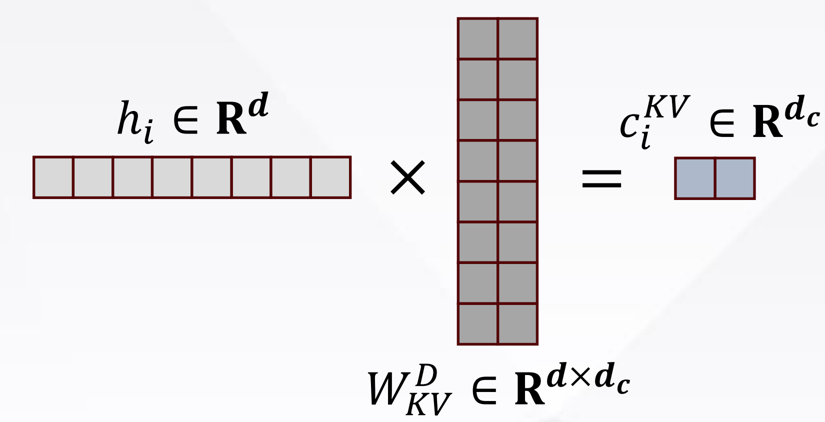
Hidden States

Key (Value)

$W_{KV}^D \in \mathbf{R}^{d \times d_c}$ × $W_K^U, W_V^U \in \mathbf{R}^{d_c \times n_h d_h}$

如此一来，对于KV Cache，只用存储

compressed latent vector $c_i^{KV} = h_i W_{KV}^D$

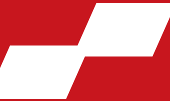


- W_K^U与W_Q合并, W_V^U与W_O合并
- KV cache per token： $\boldsymbol{d_c l}$, $d_c \ll n_h d_h$

Liu A, Feng B, Wang B, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model[J]. arXiv preprint arXiv:2405.04434, 2024.

SJTU

Multi-head Latent Attention (MLA)



- ……但是，这样的MLA与RoPE不兼容

RoPE的旋转位置编码矩阵： R_i

$$q_i = h_i W_Q R_i, \quad k_j = h_j W_K R_j = c_j^{KV} W_K^U R_j$$

$$q_i k_j^T = (h_i W_Q) \boldsymbol{R_{i-j}} (c_j^{KV} W_K^U)^T$$

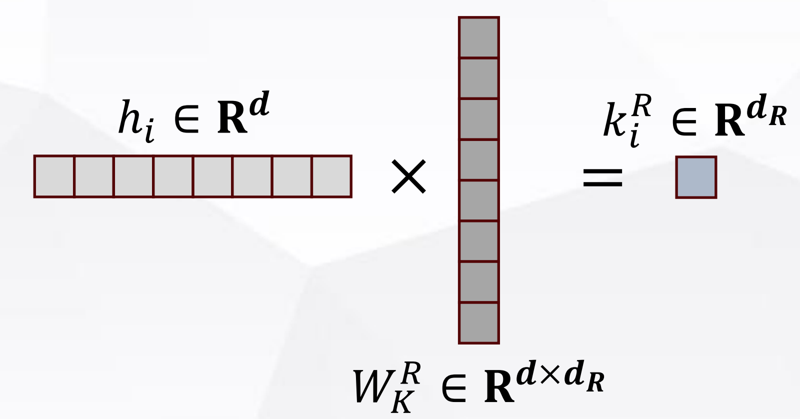
R_{i-j}的存在，使得W_K^U无法与W_Q合并

- 将位置编码**解耦**到新的维度上$k_i^R \in \mathbf{R}^{d_R}$：

$$k_i^R = RoPE(h_i W_K^R), \quad W_K^R \in \mathbf{R}^{d \times d_R}$$

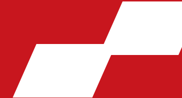
$$\mathrm{k}_i^{(s)} = \left[k_i^{C^{(s)}}; k_i^R\right]$$

- 所有head共享位置信息
- Cache per token： $(\boldsymbol{d_c} + \boldsymbol{d_R})\boldsymbol{l}$



Liu A, Feng B, Wang B, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model[J]. arXiv preprint arXiv:2405.04434, 2024.

Multi-head Latent Attention (MLA)



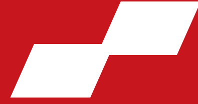
- $d_c = 4d_h$
- $d_R = \frac{d_h}{2}$

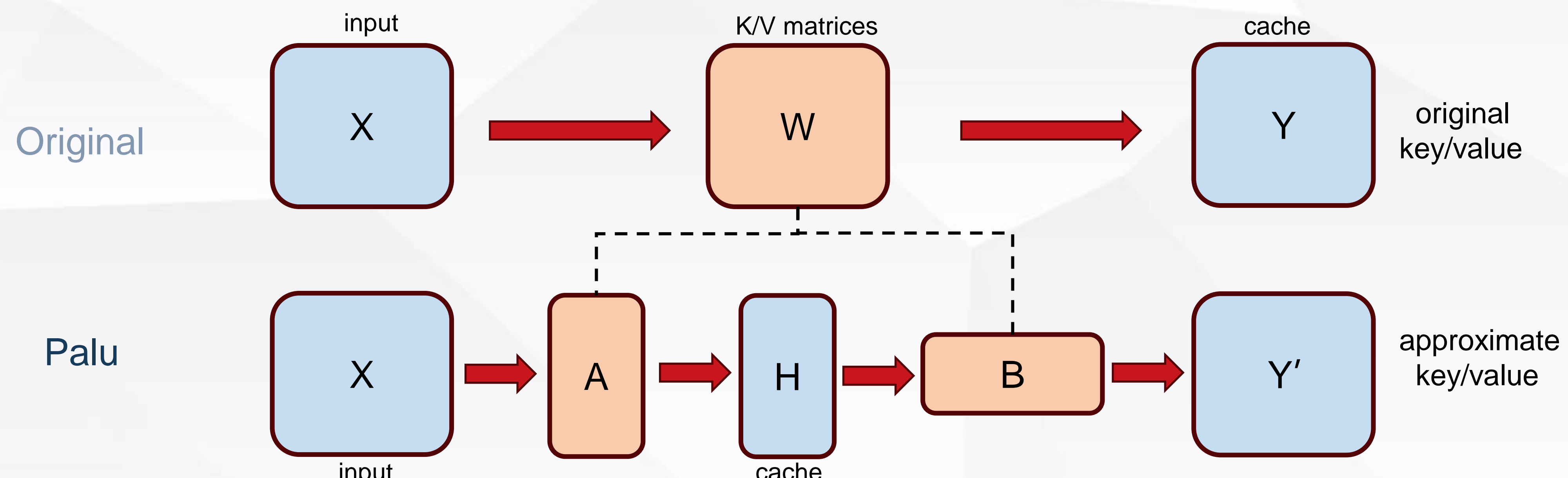
Attention Mechanism	KV Cache per Token (# Element)	Capability
Multi-Head Attention (MHA)	$2n_h d_h l$	Strong
Grouped-Query Attention (GQA)	$2n_g d_h l$	Moderate
Multi-Query Attention (MQA)	$2d_h l$	Weak
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2}d_h l$	Stronger

Benchmark (Metric)	# Shots	Small MoE w/ MHA	Small MoE w/ MLA	Large MoE w/ MHA	Large MoE w/ MLA
# Activated Params	-	2.5B	2.4B	25.0B	21.5B
# Total Params	-	15.8B	15.7B	250.8B	247.4B
KV Cache per Token (# Element)	-	110.6K	15.6K	860.2K	34.6K
BBH (EM)	3-shot	37.9	**39.0**	46.6	**50.7**
MMLU (Acc.)	5-shot	48.7	**50.0**	57.5	**59.0**
C-Eval (Acc.)	5-shot	**51.6**	50.9	57.9	**59.2**
CMMLU (Acc.)	5-shot	52.3	**53.4**	60.7	**62.5**

Liu A, Feng B, Wang B, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model[J]. arXiv preprint arXiv:2405.04434, 2024.

Palu





$$Y = XW$$

$$W = U\Sigma V^T \approx U_r \Sigma_r V_r^T = \left(U_r \sqrt{\Sigma_r}\right)\left(\sqrt{\Sigma_r} V_r^T\right) = AB$$

$$XA = H \rightarrow cache$$

$$cache \rightarrow HB = Y'$$

- Palu的总体思想与MLA类似。但是Palu没有合并K和V的cache，且通过SVD分解来实现对训练好的原attention参数进行低秩逼近。
- 线性映射W低秩分解为A和B，输入X经A向下投影成H，存储到kv cache中。
- 取出H，经B向上投影（也称为reconstruction），获取原始KV的近似

Chang, Chi-Chih, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. "Palu: Compressing KV-Cache with Low-Rank Projection." *arXiv preprint arXiv:2407.21118* (2024).

Palu

Input
W^q
q_t
RoPE
Low-Rank Projection
A^k
h_t^{key}
h_{t-1}^{key}
h_1^{key}
Reconstruct on-the-fly
B^k
K_t
RoPE
$q_t@K^T$ (GEMV)
Score
Softmax
a@H (GEMV)
B^v
Absorb
W^o
Offline Fusion
B^vW^o (Fused)
Output
Low-Rank Projection
A^v
h_t^{val}
h_{t-1}^{val}
h_1^{val}
Low-Rank Key (Cached)
Original Key (Not cached)
Low-Rank Value (Cached)

$$a_i W_i^o = (p_i V_i) W_i^o = (p_i H_i^v B_i^v) W_i^o = p_i H_i^v (B_i^v W_i^o)$$

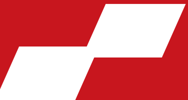
- Value-reconstruction可以fusion到W^o，减少了W^o的参数量，可以减少矩阵乘法计算，进一步加速inference。

- Key-reconstruction过程想要融入到W^q中，会受RoPE的非线性影响无法实现。因此总体reconstruction开销只能减少一部分，无法真正消除。

SJTU

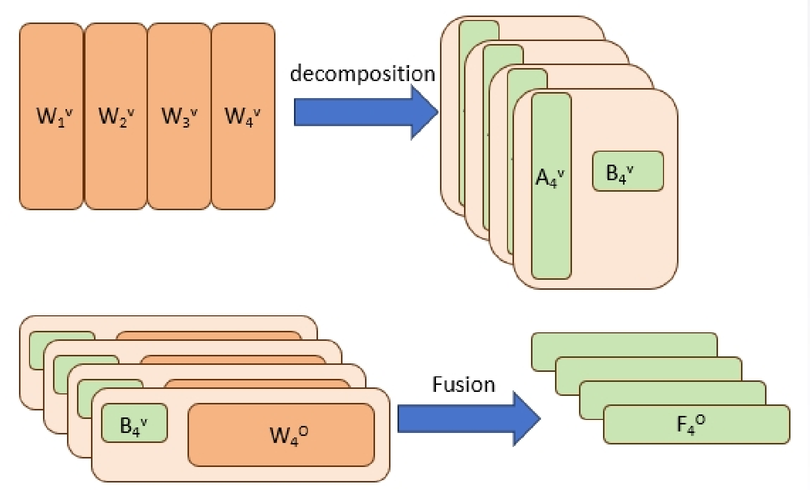
Chang, Chi-Chih, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. "Palu: Compressing KV-Cache with Low-Rank Projection." *arXiv preprint arXiv:2407.21118* (2024).

Palu的低秩分解粒度



Multi-Head Low Rank Decomposition

Low computation cost and weight memory



Lost common feature and Bad accuracy

$$W_{\text{joint}} = (W_1, W_2, ..., W_n)$$

$$Find(A_1, A_2, ..., A_n, B_1, B_2, ..., B_n)$$

Let

$$W_1 \approx A_1 B_1$$

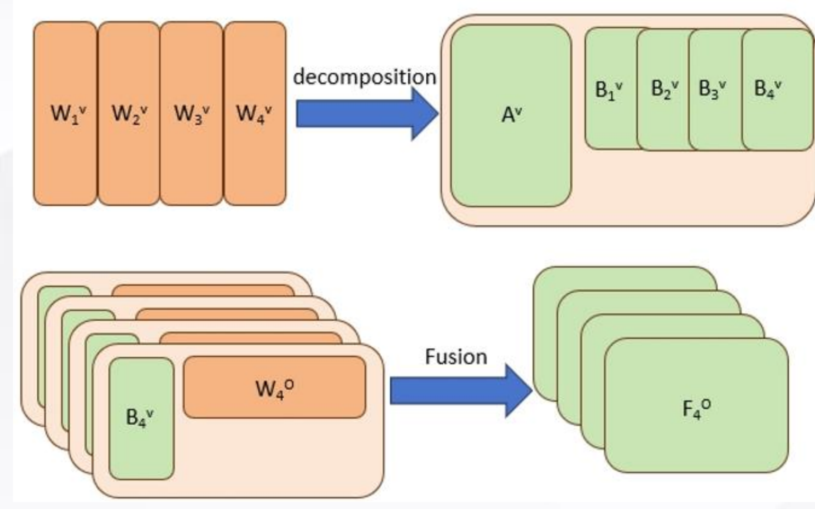
$$W_2 \approx A_2 B_2$$

$$...$$

$$W_n \approx A_n B_n$$

Joint-Head Low Rank Decomposition

High computation cost and weight



Preserve common feature and Good accuracy

$$W_{\text{joint}} = (W_1, W_2, ..., W_n)$$

$$Find(A, B_1, B_2, ..., B_n)$$

Let

$$W_1 \approx AB_1$$

$$W_2 \approx AB_2$$

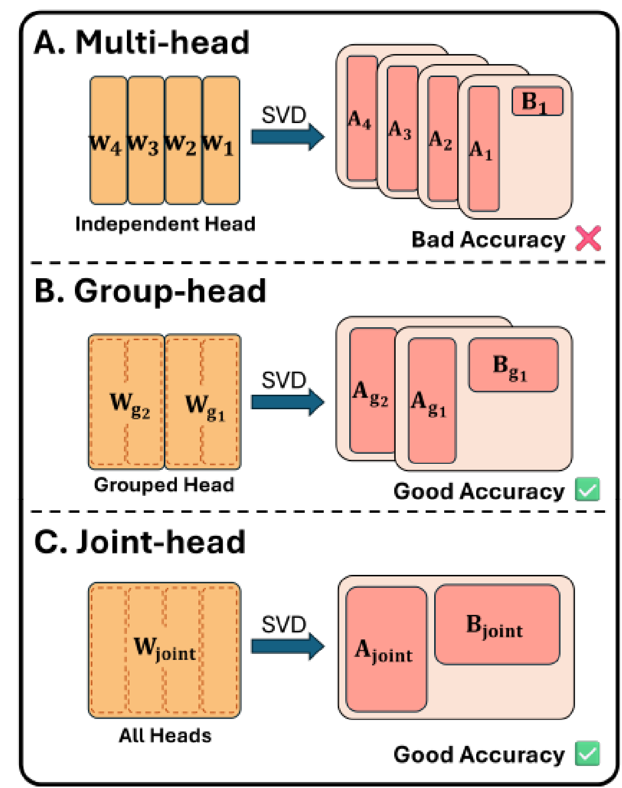
$$...$$

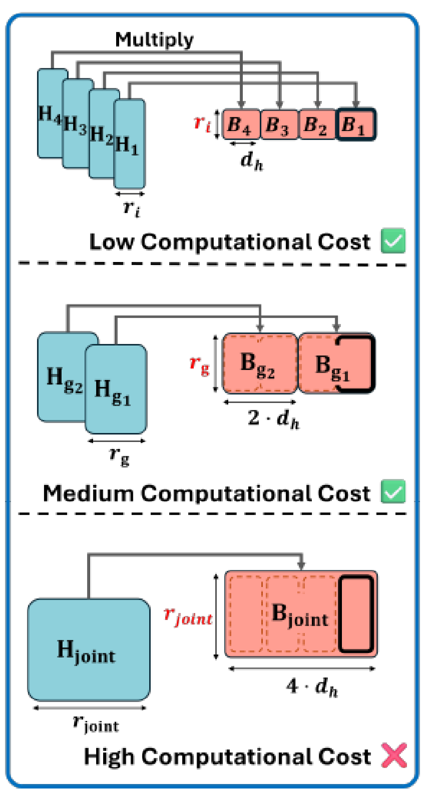
$$W_n \approx AB_n$$

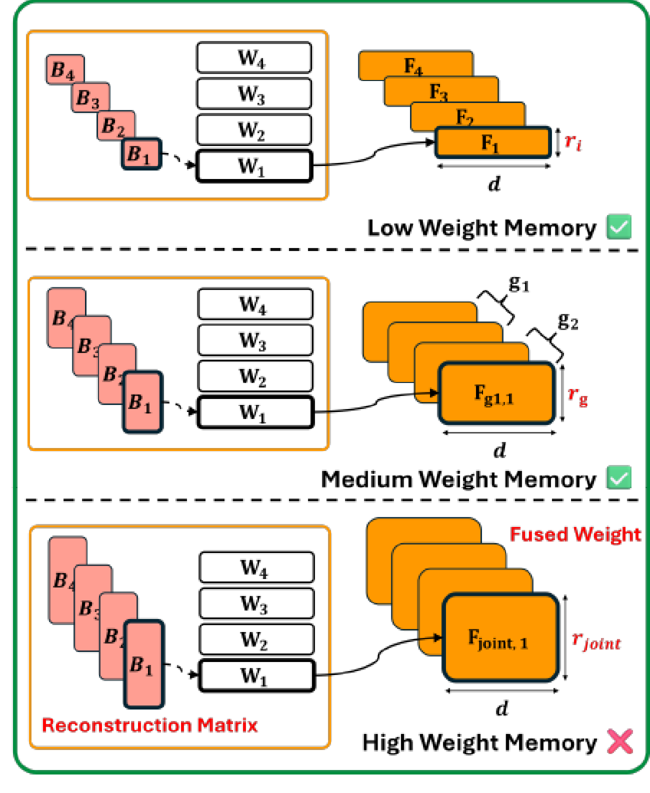
SJTU

Chang, Chi-Chih, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. "Palu: Compressing KV-Cache with Low-Rank Projection." *arXiv preprint arXiv:2407.21118* (2024).

Palu的低秩分解粒度



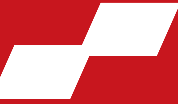




为了平衡精度损失与重建开销，采取**Group-Head Low Rank Decomposition (G-LRD)**将head分组，并以组为单位进行低秩分解，既可以利用组中的共同特征，又不会引入太多的重建开销

Chang, Chi-Chih, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. "Palu: Compressing KV-Cache with Low-Rank Projection." *arXiv preprint arXiv:2407.21118* (2024).

Palu确定低秩维度的方法



- 为了实现最优的分解效果，对于重要性高的参数矩阵，保留较高的秩，对重要性低的参数矩阵，保留较低的秩。
- 使用**Fisher Information**来衡量参数矩阵的重要性

$$I_\omega = \frac{1}{|D|}\sum_{i=1}^{|D|}\left(\frac{\partial}{\partial\omega}L(d_i;\omega)\right)^2$$

- 模型中每个参数的Fisher Information 累积了训练数据 di∈D 上的平方梯度

$$I_{W(g_i)} = \frac{1}{mn}\sum_{i=0,j=0}^{i=m,j=n} I_{W(gi,i,j)}$$

- 计算W_k与W_v中每个group的Fisher Information均值

$$R_{W(gi)} = R_t \cdot I_{W(g_i)} / \sum_{g_i} R_{W(g_i)}$$

- 根据group的Fisher Information占模型整体的比例分配该group保留的秩

Chang, Chi-Chih, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. "Palu: Compressing KV-Cache with Low-Rank Projection." *arXiv preprint arXiv:2407.21118* (2024).

SJTU

Kernal-based Attention：Performer

- Standard attention module can be represented as

$$Attention(Q, K, V) = AV = softmax(QK^T)V$$

- If we can decompose A as $A = Q'(K')^T$, the attention module can be rewritten as

$$Attention(Q, K, V) = AV = Q'(K')^T V = Q'((K')^T V)$$

batch dim

$O(L^2 d)$ $O(Lrd)$ $O(Lrd)$

$L \times L$ × $L \times d$ $\stackrel{\mathbb{E}}{=}$ $L \times r$ × $r \times L$ × $L \times d$

$(\mathbf{K}')^\top$

A attention mechanism **V** **Q′** **V**

Kernal-based Attention： Performer

- Find function ϕ

$$\phi(K) = K'$$

$$\phi(Q) = Q'$$

O(Lrd)

O(Lrd)

$$Attention(Q, K, V) = \phi(Q)(\phi(K)^T \phi(V))$$

$O(L^2 d)$

batch dim

$L \times L$ × $L \times d$ $\mathbb{E}$ = $L \times r$ × $r \times L$ × $L \times d$

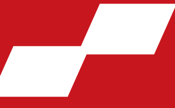
$O(Lrd)$ $O(Lrd)$

$(\mathbf{K}')^\top$

$\mathbf{A}$ attention mechanism $\mathbf{V}$ $\mathbf{Q}'$ $\mathbf{V}$

SJTU

Kernal-based Attention：Performer



Find function ϕ
that can **approximate** to Softmax

$$Softmax(Q, K) \approx \phi(Q)\phi(K)$$

or **replace** it

$$Softmax(Q, K) \Rrightarrow \phi(Q)\phi(K)$$

Kernal-based Attention： Performer

- Decompose softmax calculation
- Performers find a function to approximate by independent random samples ω from Standard Gaussian distribution

$$e^{q\cdot k} = \phi(q)\cdot\phi(k) = \tilde{q}\cdot\tilde{k}$$

$$e^{\mathbf{q}\cdot\mathbf{k}} = \mathbb{E}_{\omega\sim\mathcal{N}(\omega;0,1_d)}\left[e^{\omega\cdot\mathbf{q}-\|\mathbf{q}\|^2/2}\times e^{\omega\cdot\mathbf{k}-\|\mathbf{k}\|^2/2}\right] \approx \frac{1}{\sqrt{m}}\begin{pmatrix} e^{\omega_1\cdot\mathbf{q}-\|\mathbf{q}\|^2/2} \\ e^{\omega_2\cdot\mathbf{q}-\|\mathbf{q}\|^2/2} \\ \vdots \\ e^{\omega_m\cdot\mathbf{q}-\|\mathbf{q}\|^2/2} \end{pmatrix}\cdot\frac{1}{\sqrt{m}}\begin{pmatrix} e^{\omega_1\cdot\mathbf{k}-\|\mathbf{k}\|^2/2} \\ e^{\omega_2\cdot\mathbf{k}-\|\mathbf{k}\|^2/2} \\ \vdots \\ e^{\omega_m\cdot\mathbf{k}-\|\mathbf{k}\|^2/2} \end{pmatrix}$$

Expect approximate to Softmax when m -> ∞

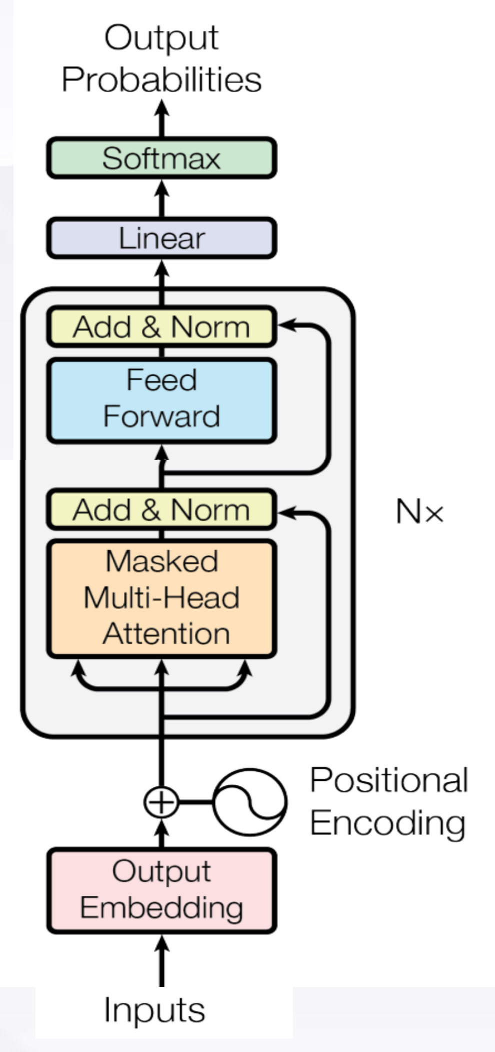
$\tilde{q}$ $\tilde{k}$

where $\tilde{q},\tilde{k}\in\mathbb{R}^m \quad q,k\in\mathbb{R}^d$

Weakness: m need to be large

Choromanski, Krzysztof Marcin, et al. "Rethinking Attention with Performers." International Conference on Learning Representations. 2020.

目录



1 • Attention机制层面

2 • 逐层KV Cache压缩层面

3 • 层间KV Cache压缩层面

4 • 宏观计算架构适配层面

SJTU

目录

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Inputs

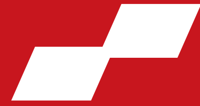
2

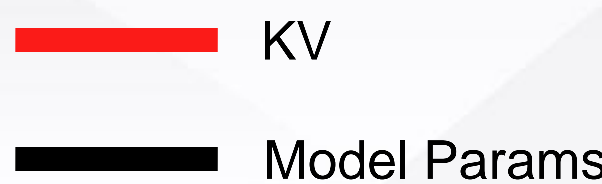
• 逐层KV Cache压缩层面

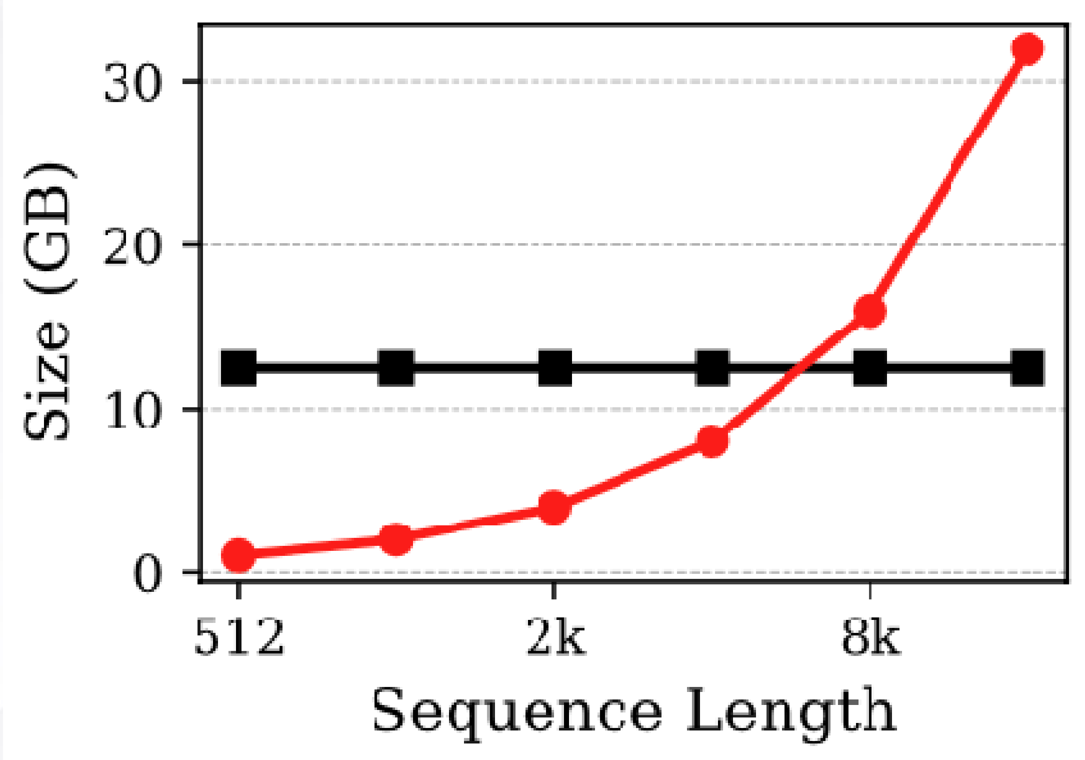
这一大类方法，主要在对每一层的KV Cache序列维度的长度进行压缩。

- Streaming LLM、LM-Infinite
- InfLLM
- H2O、Scissorhands、TOVA
- TreeKV
- SnapKV
- PyramidKV

KV Cache是长文本处理中限制模型效率的瓶颈

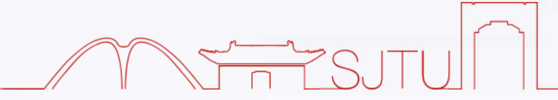




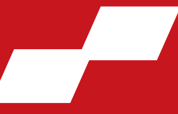


Keyformer. Adnan et al.

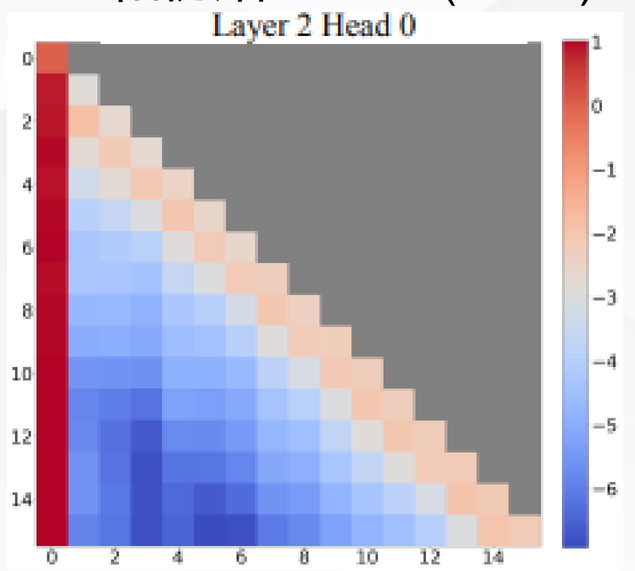
- KV Cache占用的显存空间随着序列长度的增加以平方复杂度增长
- 前面我们讲到的的方法有试图去降低K或者V的维度，这些方法对不同长度的序列加速效果是一样的。
- 如果能降低KV Cache的序列长度，在超长文本上将能获得二次方的加速回报。



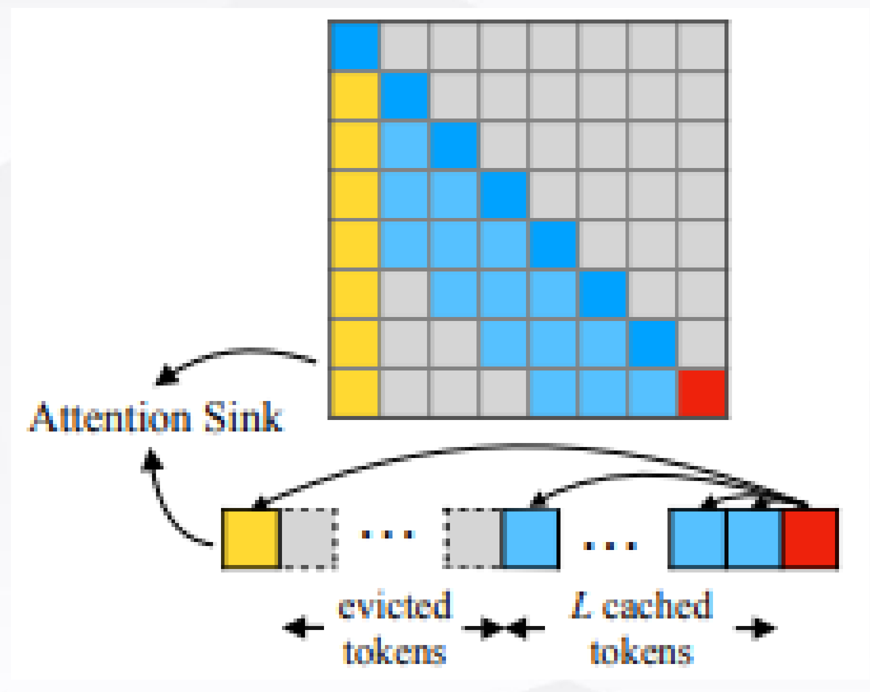
Position-Based: StreamingLLM



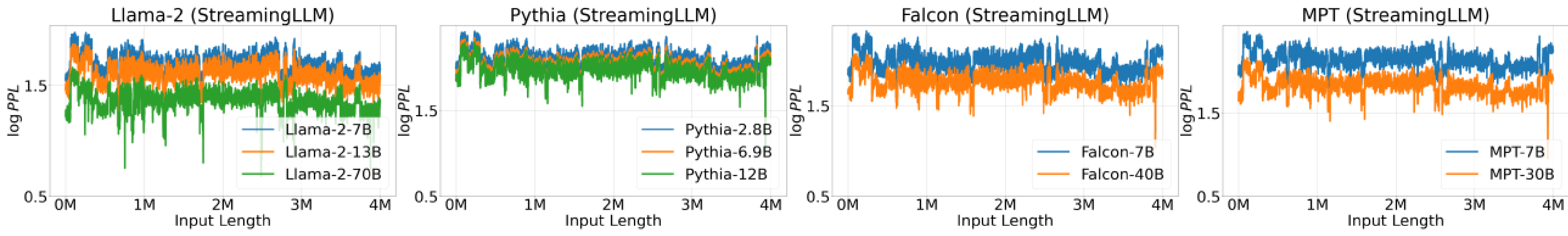
在attention maps上的发现：注意力分数主要集中在最近tokens和初始 tokens(sinks)上。



只在cache中保留这些总被注意到的tokens

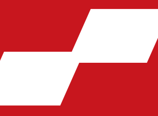


能够做到在4M长文本上PPL稳定：

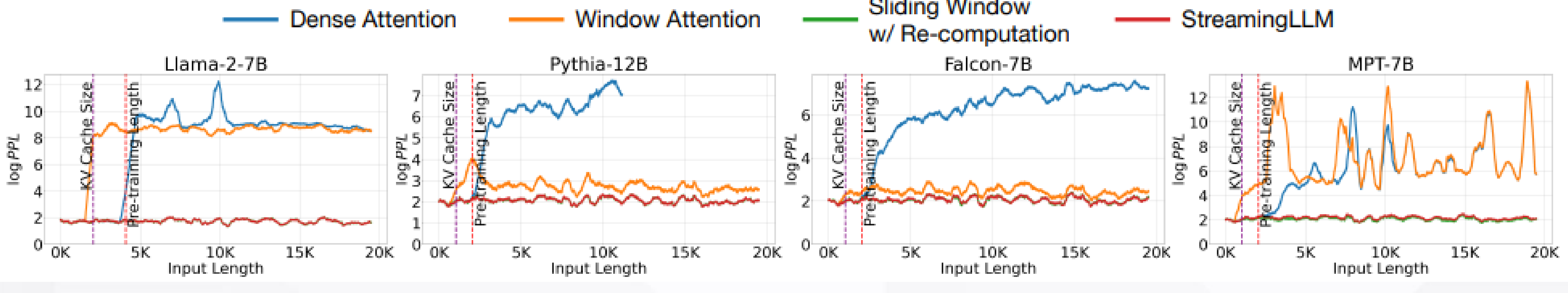


Efficient Streaming Language Models with Attention Sinks, Xiao et al, ICLR 2024

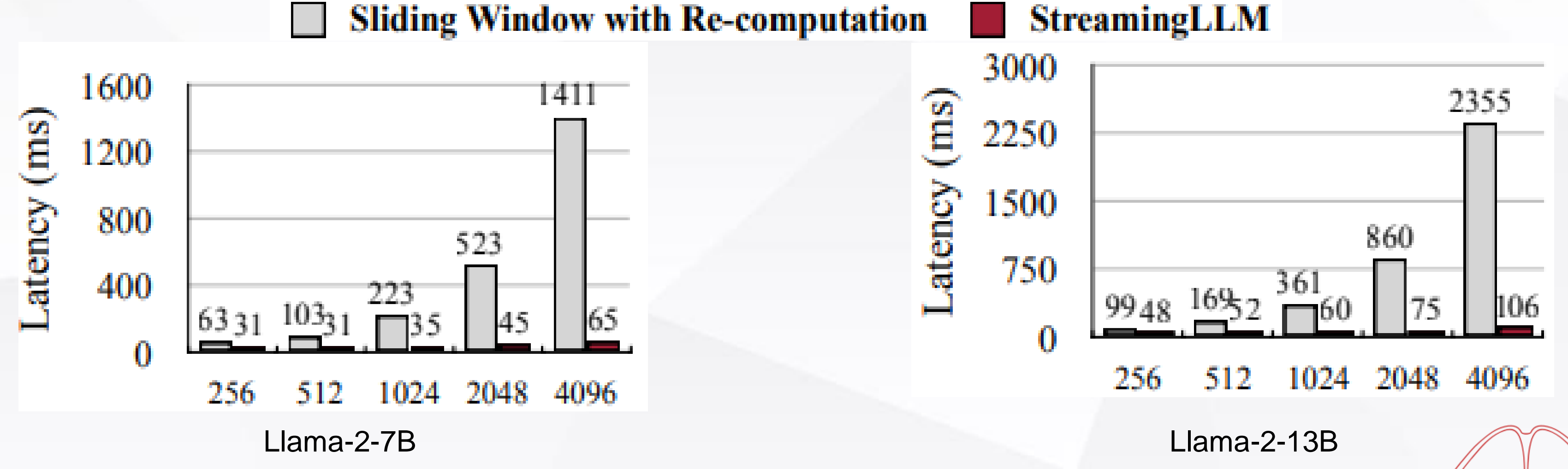
Position-Based: StreamingLLM



相较于已有的Window Attention方法： 超预训练长度后，在ppl上表现更好



相较于已有的Sliding Window w/Re-computation方法： 推理速度更快

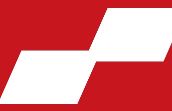


Llama-2-7B

Llama-2-13B

Efficient Streaming Language Models with Attention Sinks, Xiao et al, ICLR 2024

Position-Based: StreamingLLM



原理分析：

sinks起到什么作用?

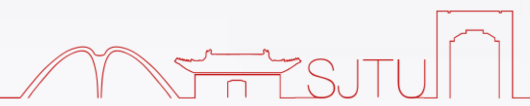
- 蕴含重要语义信息? ✗
- 收集多余attention?

把开头4个token换成没有语义信息的\n，一样能降低ppl

Llama-2-13B	PPL (↓)
0 + 1024 (Window)	5158.07
4 + 1020	5.40
4"\n"+1020	5.60

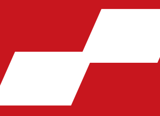
为什么是初始几个tokens作为sinks?

由于自回归模型的顺序性，训练时初始的token是全序列可见的



Efficient Streaming Language Models with Attention Sinks, Xiao et al, ICLR 2024

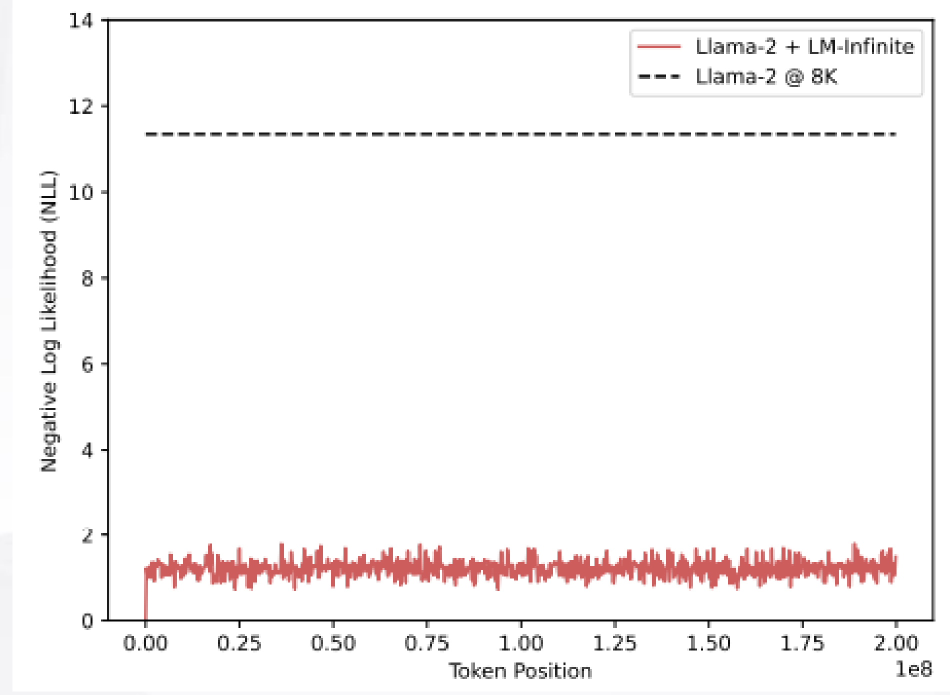
Position-Based: LM-Infinite

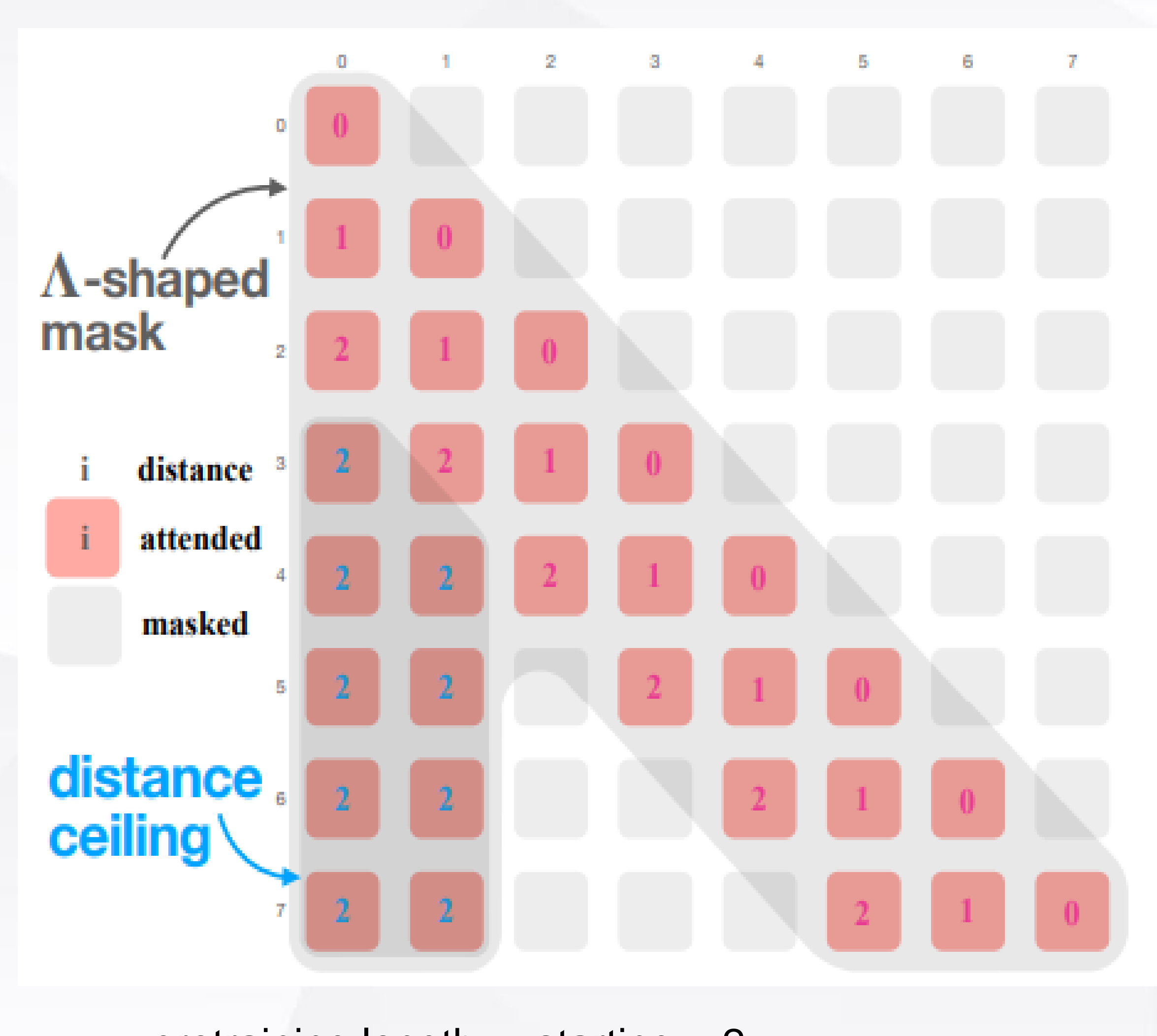


压缩的**思路和StreamLLM一样**，保留最近tokens和初始 tokens。区别在于

- LM-Infinite的Recent取预训练长度。
- 初始token的位置编码和Recent左端一致。且尝试了5~100范围内的初始token数目。

能在两亿文本上保持PPL稳定：

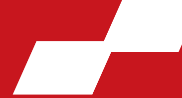




pretraining length = starting = 2

LM-Infinite: Zero-Shot Extreme Length Generalization for Large Language Models, Han et al, NAACL 2024

Position-Based: LM-Infinite



LM-Infinite不用训练，但能比在更长文本长度作训练的模型有更好的PPL。

Model	Train Leng.	ArXiv 2K	ArXiv 4K	ArXiv 8K	ArXiv 16K	ArXiv 32K	OpenWebText2 2K	OpenWebText2 4K	OpenWebText2 8K	OpenWebText2 16K
Long-context Training/Finetuning										
Sandwich	512	5.0	5.2	5.3	-	-	23.3	23.8	24.7	-
XPos	1K	21.6	20.7	-	-	-	-	-	-	-
LongLLaMA	8K	8.2	7.4	-	6.9	-	-	-	-	-
MPT-7B-SW	65K	6.5	5.4	4.3	4.4	3.6	9.8	10.9	6.6	**5.1**
Vanilla										
MPT-7B	4K	5.5	2.5×10^2	1.1×10^3	1.7×10^3	1.6×10^3	8.3	1.3×10^2	1.9×10^2	1.3×10^2
LLaMA	2K	3.8	1.0×10^4	6.0×10^4	6.8×10^4	4.9×10^4	6.2	6.6×10^3	4.6×10^5	4.4×10^4
GPT-J-6B	2K	3.9	1.3×10^3	1.0×10^3	1.6×10^3	2.8×10^2	8.8	7.5×10^2	1.3×10^3	1.8×10^3
Llama-2	4K	**3.4**	3.8	8.5×10^3	NaN	NaN	6.2	5.8	6.5×10^3	NaN
LM-Infinite										
MPT-7B	4K	5.7	6.8	5.8	6.0	4.6	8.5	12.2	8.5	8.9
LLaMA	2K	4.4	4.5	3.7	4.2	**1.0**	6.3	6.1	9.5	7.0
GPT-J-6B	2K	3.8	**3.1**	**3.0**	**3.1**	2.1	8.8	8.5	**6.5**	7.4
Llama-2	4K	4.3	3.6	3.3	4.2	6.5	**6.1**	**5.3**	8.3	8.2

SJTU

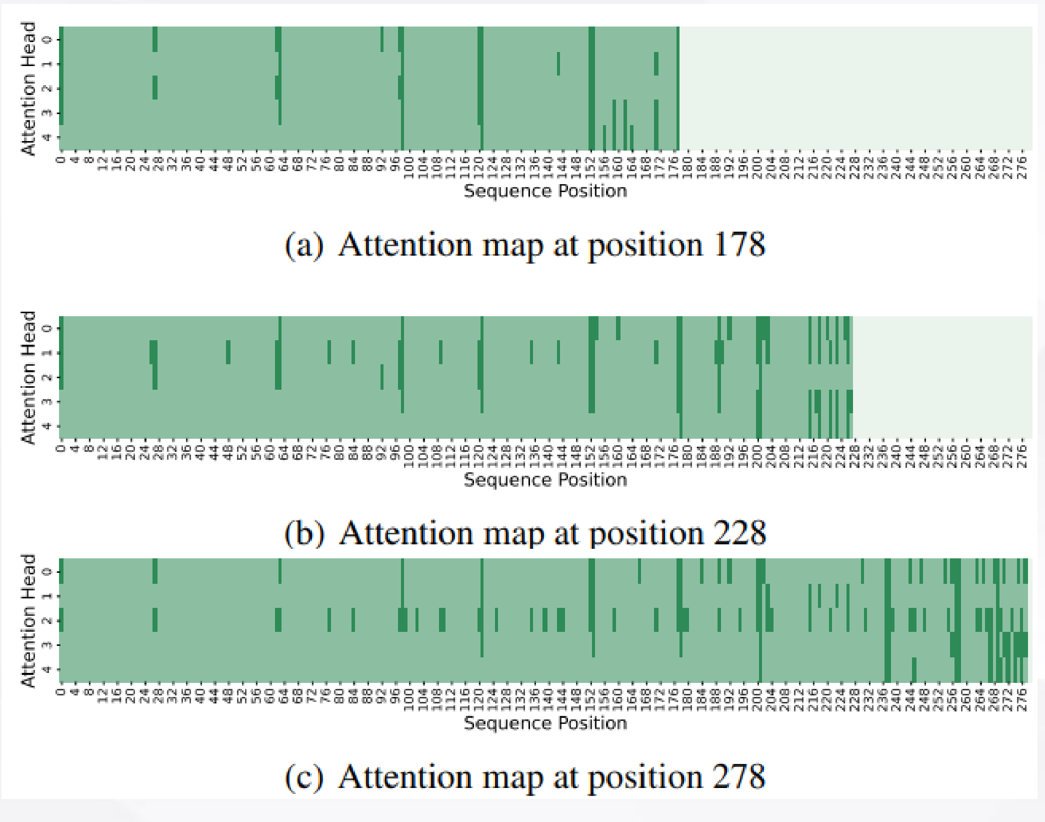
LM-Infinite: Zero-Shot Extreme Length Generalization for Large Language Models, Han et al, NAACL 2024

Content-Based

仅靠位置信息是否会遗漏重要信息?

观察：

- 在LLMs的生成过程中，不同的query会对固定一小部分token保持高相关度。



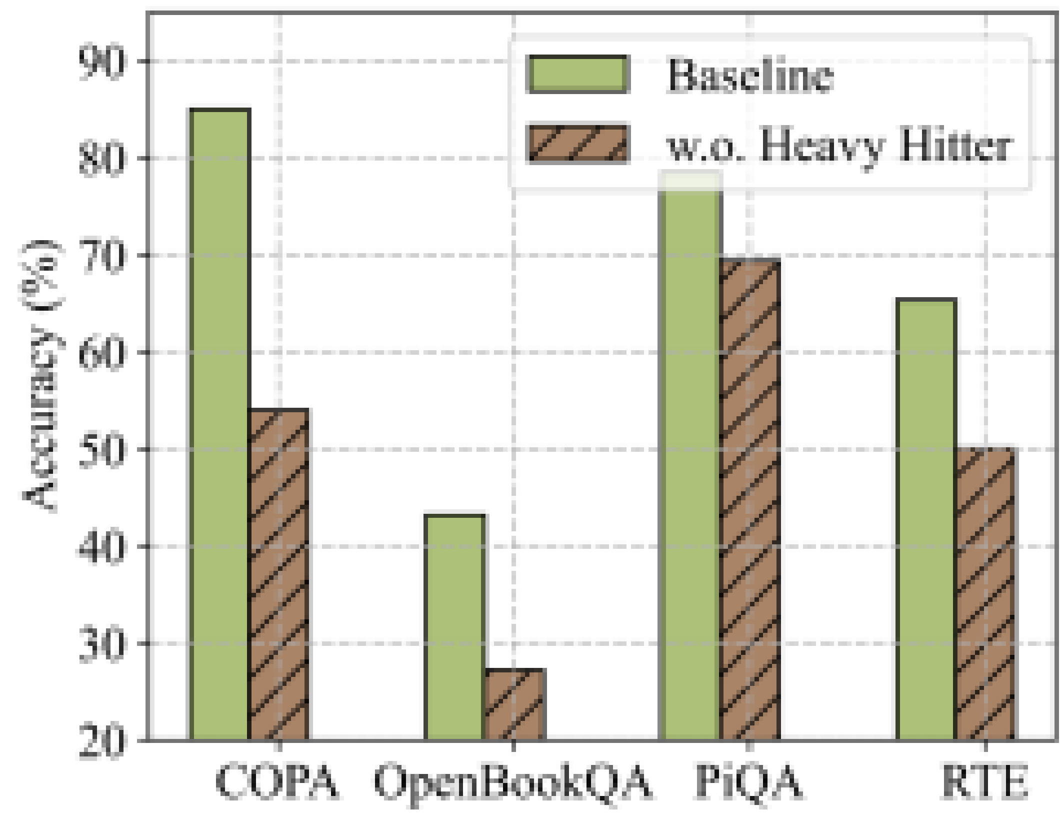
(a) Attention map at position 178

(b) Attention map at position 228

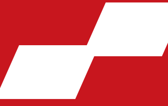
(c) Attention map at position 278

由于这些重要tokens位置因sample而异，需要借助Content-based的方法才能较好地找到它们。

- 这部分token对模型表现很重要。



Content-Based: InfLLM

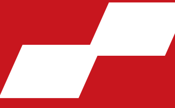


InfLLM将全部过去KV分块存入主存，根据注意力分数取回重要块至显存加入KV Cache。

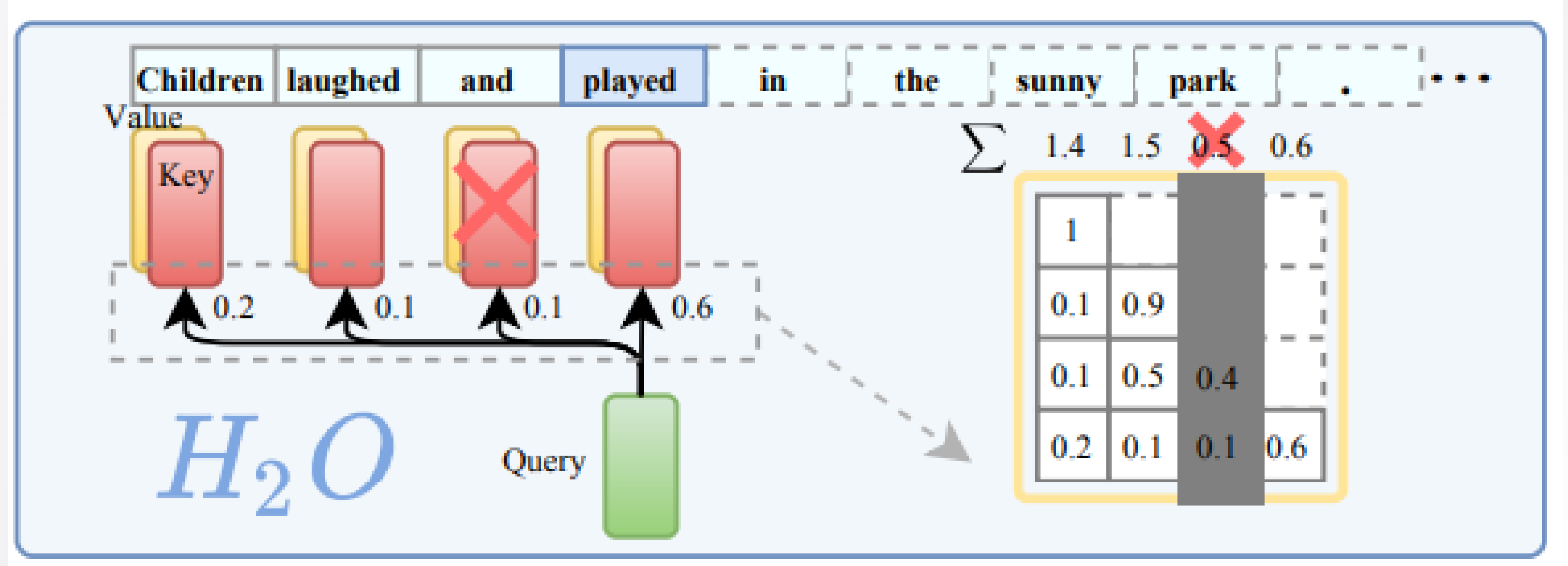
Memory Units
Representative Tokens
Context Memory
Lookup
Current Key-Value Cache
LLM
Inputs
Past Key-Value
Initial Tokens
Evicted Tokens
Local Tokens
Current Tokens

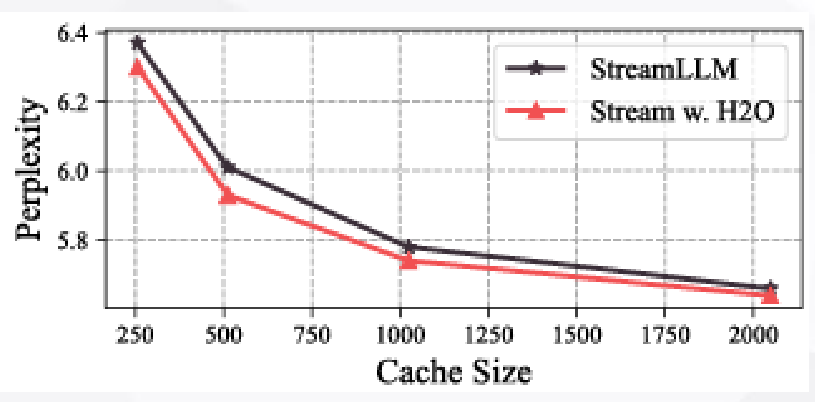
SJTU

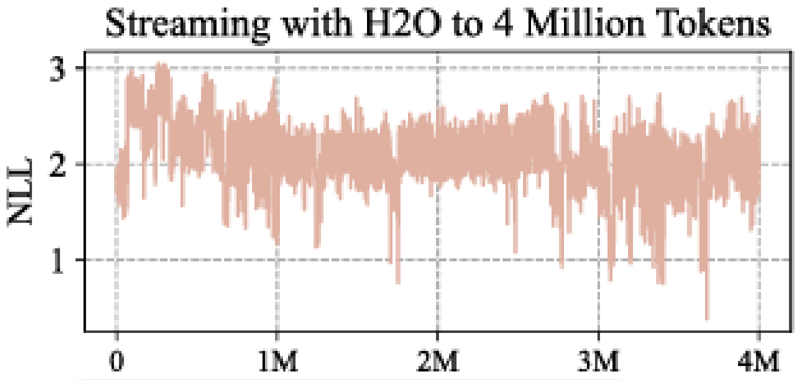
Content-Based: H2O



H2O在KV Cache中保留了：
- 最近tokens
- Heavy Hitters: 累积注意力分数最大的tokens







能在2M文本上保持PPL稳定，PG19第一个样本上PPL优于StreamingLLM

H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, Zhang et al, NeurIPS 2023

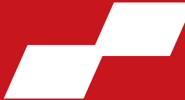
Content-Based: H2O

与只用Local的方法对比：

XSUM, LLaMA-7B
XSUM, LLaMA-13B
XSUM, LLaMA-30B
XSUM, GPT-NeoX-20B
CNN/Daily Mail, LLaMA-7B
CNN/Daily Mail, GPT-NeoX-20B
OpenBookQA, OPT-30B
OpenBookQA, OPT-66B
COPA, OPT-30B
COPA, OPT-66B
MathQA, OPT-30B
RTE, OPT-66B

H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, Zhang et al, NeurIPS 2023

Content-Based: H2O



T4 GPU上Generation throughput (token/s)的提升：

Seq. length	512+32		512+512		512+1024	
Model size	6.7B	30B	6.7B	30B	6.7B	30B
Accelerate	20.4 (2, G)	0.6 (8, C)	15.5 (1, G)	0.6 (8, C)	5.6 (16, C)	0.6 (8, C)
DeepSpeed	10.2 (16, C)	0.6 (4, C)	9.6 (16, C)	0.6 (4, C)	10.1 (16, C)	0.6 (4, C)
FlexGen	20.2 (2, G)	8.1 (144, C)	16.8 (1, G)	8.5 (80, C)	16.9 (1, G)	7.1 (48, C)
H_2O (20%)	35.1 (4, G)	12.7 (728, C)	51.7 (4, G)	18.83 (416, C)	52.1 (4, G)	13.82 (264, C)

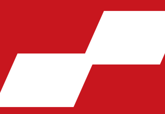
A100 GPU上latency和throughput的提升：

Seq. length	Model size	Batch size	Metric	FlexGen	H_2O (20%)
7000+1024	30B	1	latency (s)	57.0	50.4
5000+5000	13B	4	latency (s)	214.2	155.4
2048+2048	6.7B	24	latency (s)	99.5	53.5
2048+2048	6.7B	24	throughput (token/s)	494.1	918.9
2048+2048	6.7B	64	throughput (token/s)	OOM	1161.0

SJTU

H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, Zhang et al, NeurIPS 2023

Content-Based: Scissorhands



Scissorhands在KV Cache中保留了：

- 最近tokens
- 累积**二值化**注意力分数最大的tokens

可以做到在20%压缩率下不影响模型表现。

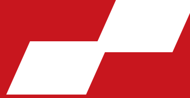
(a) Language Modeling

(b) OPT-6B Five shot

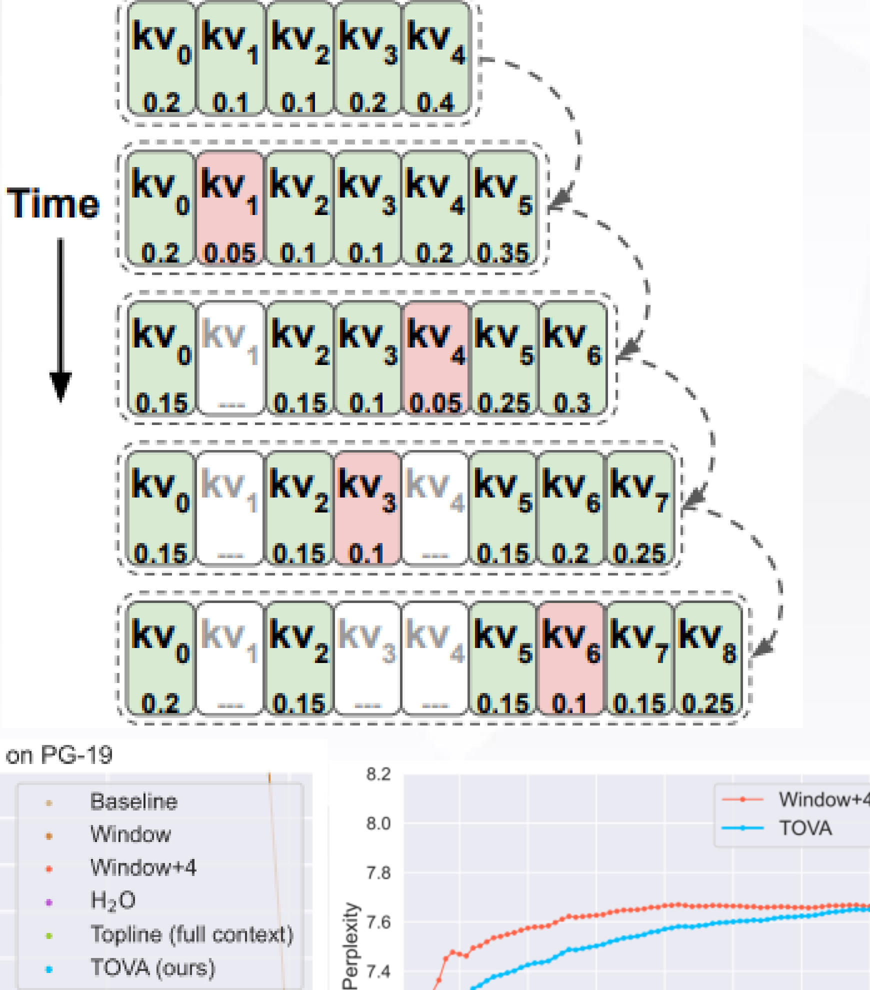
(c) OPT-13B Five shot

(d) OPT-30B Five shot

Content-Based: TOVA

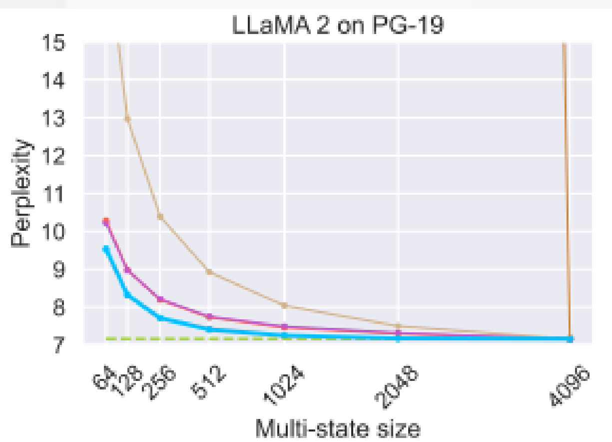


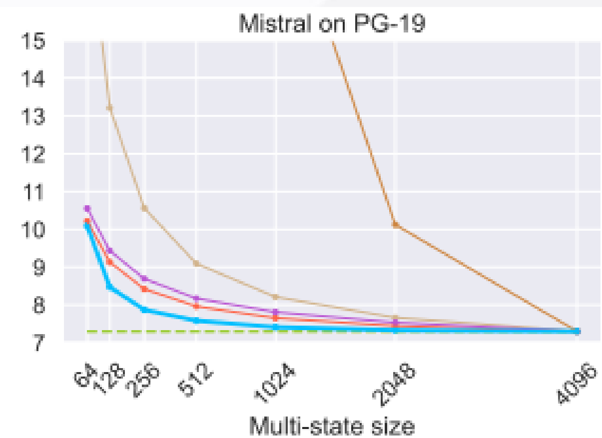
TOVA用上一个Query的注意力分数来淘汰KV对。

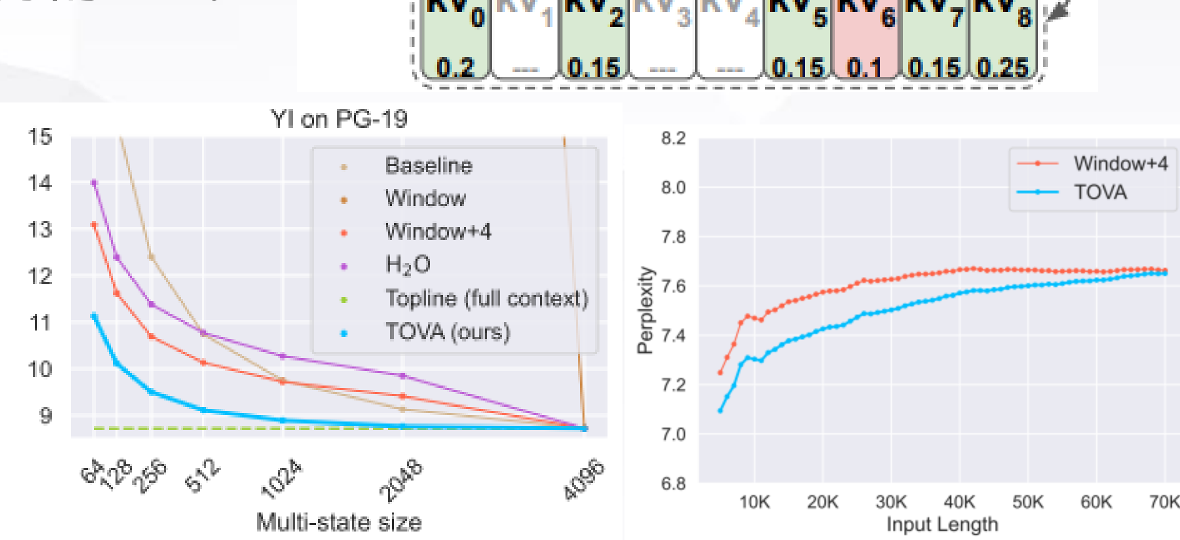


Language Modeling :

- 在PG-19(4k chunk)上相较于H2O,StreamingLLM有更好的PPL。
- 但文本更长时优势减弱。

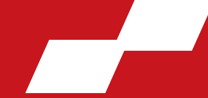






Transformers are Multi-State RNNs, Oren et al, 2024

Content-Based: TOVA



Long Range Understanding: 在SQuALITY,QASPER上相较于H2O,StreamingLLM有更好表现。

LLaMA 2-chat on SQuALITY

Mistral-Instruct on SQuALITY

neural-chat on SQuALITY

LLaMA 2-chat on QASPER

Mistral-Instruct on QASPER

neural-chat on QASPER

ROUGE

F_1

Multi-state size

256 512 1024 2048 4096

Baseline

Window+4

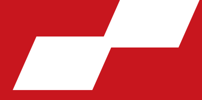
Topline (full context)

TOVA (ours)

SJTU

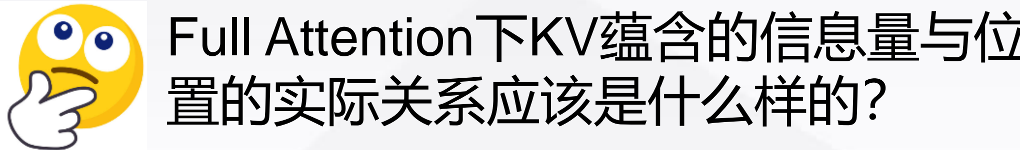
Transformers are Multi-State RNNs, Oren et al, 2024

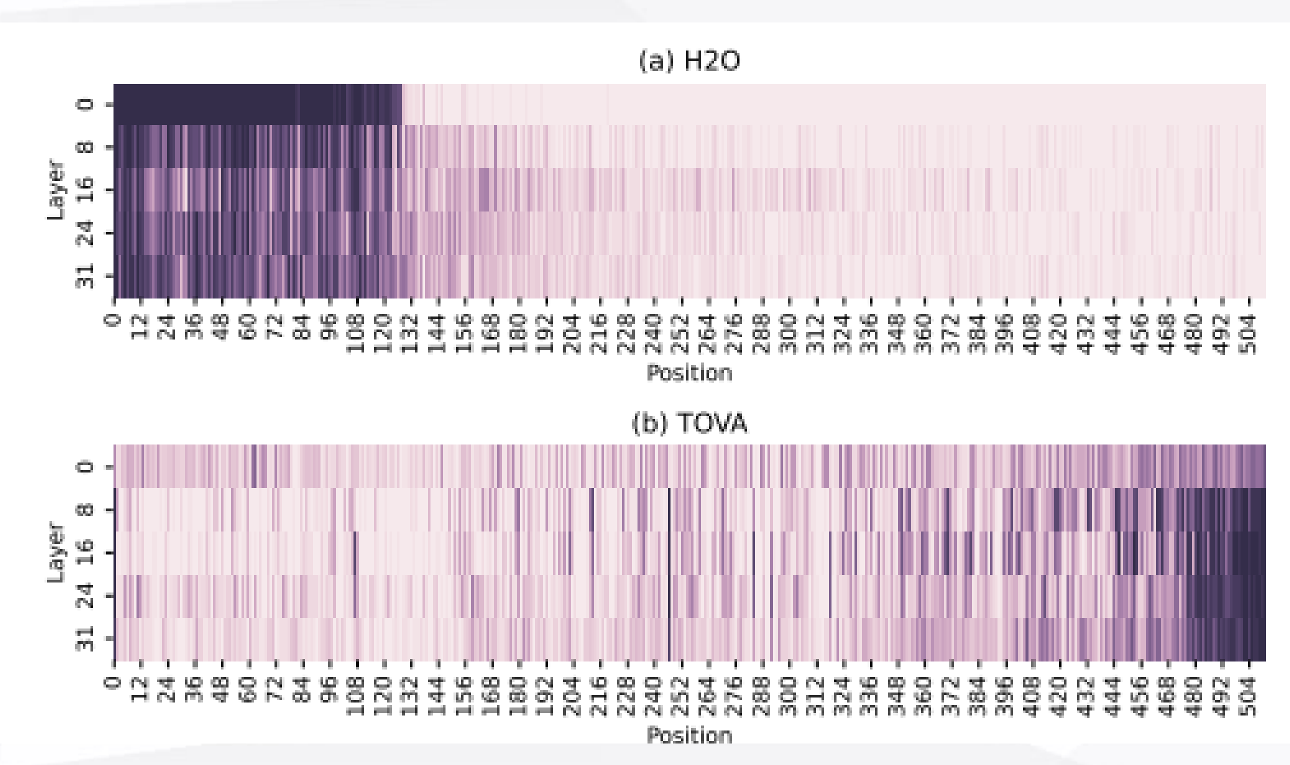
Content-Based: TreeKV

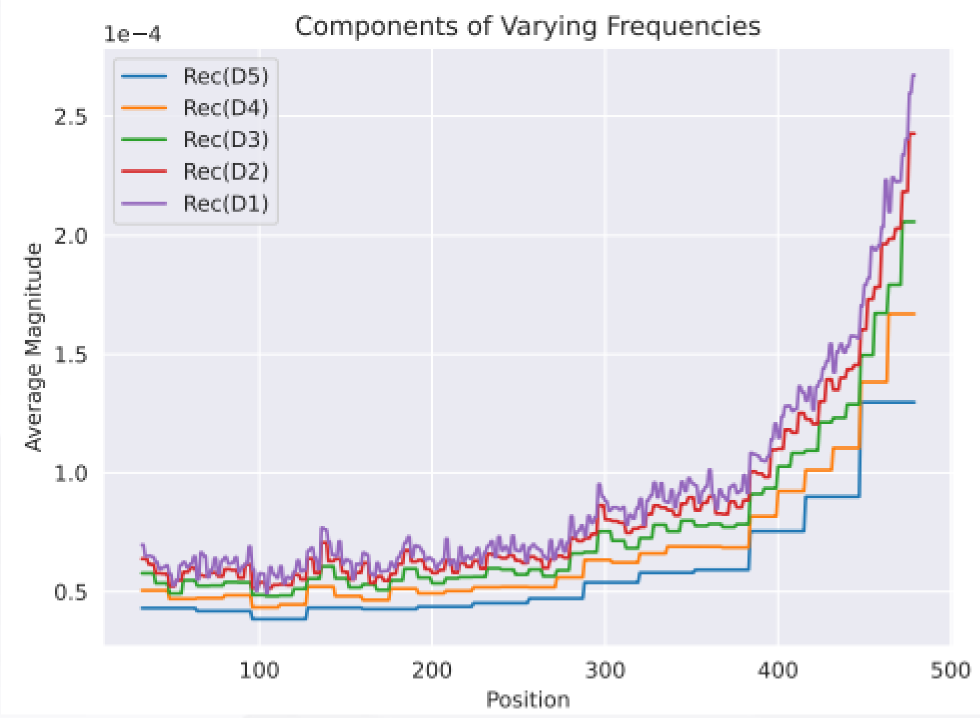


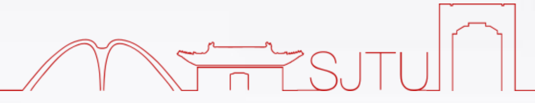
发现：

H2O和TOVA在压缩时对KV的位置有着相反的固定偏好。

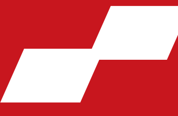








Content-Based: TreeKV



TreeKV将KV组织成树状层次结构，利用左疏右密的特性来增强缓存压缩的平滑性。

Step 4, max cache capacity 4 is reached, idx = 1

(a) H2O

(b) TOVA

(c) TreeKV

Layer

Position

The illustration of the complete eviction process at Step 17, idx = 1

tokens kept in cache

evicted tokens

cache

selection scope

sunset

sunset

sunset

casting

vibrant

sunset slowly the casting vibrant orange pink across

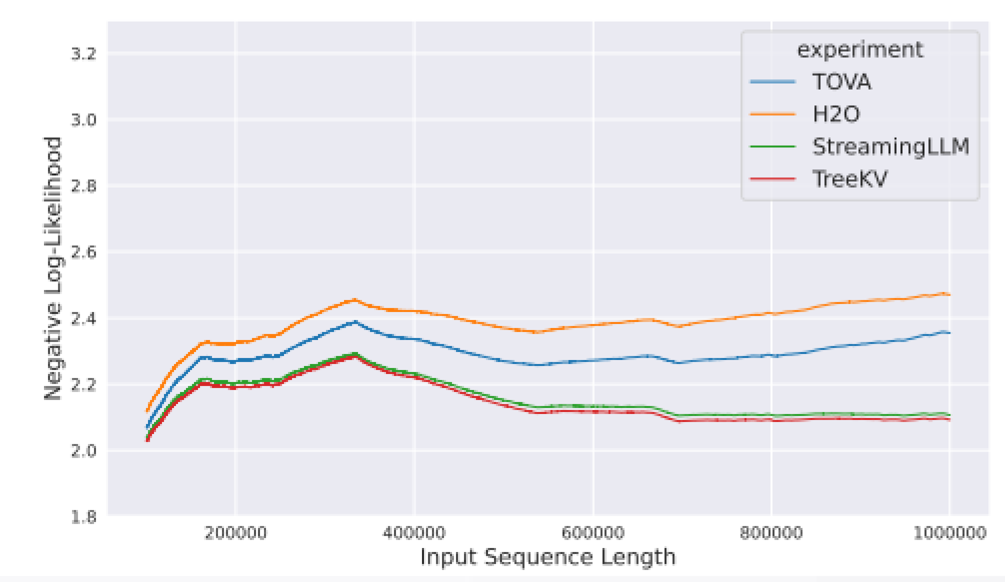
The sunset slowly over the horizon casting vibrant hues of orange and pink across the evening

new token

TreeKV: Smooth Key-Value Cache Compression with Tree Structures, He et al, 2024

Content-Based: TreeKV

- 语言建模任务上我们在两个数据集上报告了困惑度：PG19测试集和OpenWebText2。
- 我们测试了TreeKV是否能让使用4k窗口预训练的LLM能够有效地对“无限长”文本进行语言建模。右图显示了输入长度从0.1M到1M的NLL曲线。



	PG19		
Context Length	4k	8k	16k
Budget Ratio	25.0%	12.5%	6.3%
Full Attn	6.84	$> 10^3$	OOM
StreamingLLM	7.19	7.19	7.19
H2O	7.06	7.08	7.25
TOVA	**7.00**	7.06	7.15
TreeKV (ours)	7.02	**6.88**	**6.91**

	OpenWebText2		
Context Length	4k	8k	16k
Budget Ratio	25.0%	12.5%	6.3%
Full Attn	5.44	$> 10^3$	OOM
StreamingLLM	5.78	5.62	5.31
H2O	**5.60**	5.48	5.25
TOVA	5.62	5.50	5.24
TreeKV (ours)	**5.60**	**5.45**	**5.18**

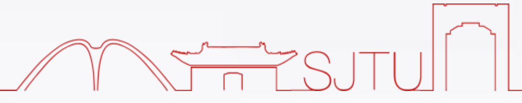
SJTU

TreeKV: Smooth Key-Value Cache Compression with Tree Structures, He et al, 2024

Content-Based: TreeKV

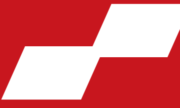
利用TreeKV的选择方式以块的方式选择KV也能在压缩prompts上取得超过H2O的效果。

Method	Len.	B.R.	Single-Document QA		Multi-Document QA		Summarization		Few-shot Learning			Synthetic		Code		Avg.
			Qasper	MF-en	HotpotQA	2WikiMQA	GovReport	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Cache Size = Full																
FullKV	0 - 4k	1	35.6	46.03	54.42	41.25	31.48	27.41	62.00	88.36	40.47	14.45	96.03	56.96	58.39	50.22
	4k - 8k	1	31.22	37.74	48.15	41.74	30.27	24.57	75.00	88.89	39.03	9.93	91.00	62.14	45.49	48.09
	8k+	1	18.96	42.03	44.42	28.64	26.22	23.15	75.00	92.30	42.87	6.99	63.00	58.68	42.93	43.47
	Avg.	1	28.59	41.93	49.00	37.21	29.32	25.04	70.67	89.85	40.79	10.46	83.34	59.26	48.94	47.26
Cache Size = 2048																
H2O	0 - 4k	50%	19.45	40.91	47.17	40.59	27.50	26.68	56.00	88.96	40.48	13.39	96.75	56.53	56.42	46.98
	4k - 8k	25%	19.77	30.17	43.39	35.66	25.25	23.05	69.00	88.78	37.40	9.39	89.50	59.53	45.47	44.33
	8k+	≤13%	8.53	38.64	44.17	28.75	23.53	21.80	64.00	92.30	42.71	7.00	63.00	61.61	42.83	41.45
	Avg.	18%	15.91	36.57	44.91	35.00	25.42	**23.84**	63.00	90.01	**40.19**	9.92	83.08	59.22	48.24	44.25
TreeKV	0 - 4k	50%	32.11	44.86	54.20	43.09	29.35	26.78	60.00	89.53	39.41	15.19	96.42	58.12	59.83	49.91
	4k - 8k	25%	29.77	33.80	47.90	38.29	26.22	22.95	75.00	88.89	36.37	9.70	90.50	60.70	47.54	46.74
	8k+	≤13%	15.37	37.89	44.25	27.01	23.30	21.47	72.00	91.82	41.33	7.74	63.00	64.12	47.42	42.82
	Avg.	18%	**25.75**	**38.85**	**48.78**	**36.13**	**26.29**	23.73	**69.00**	**90.08**	39.04	**10.88**	**83.31**	**60.98**	**51.60**	**46.49**



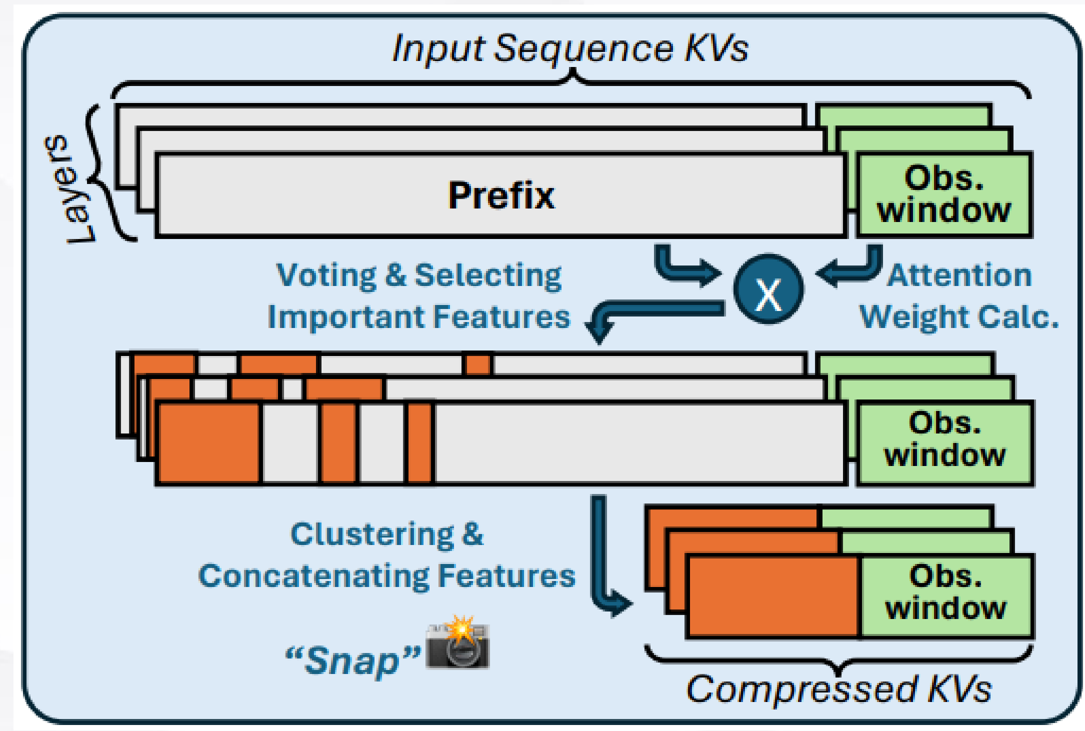
TreeKV: Smooth Key-Value Cache Compression with Tree Structures, He et al, 2024

Content-Based: SnapKV

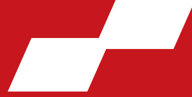


SnapKV目的仅在于对超长prompt的压缩：
- 只在prefilling阶段压缩一次。
- 生成第一个token用全长文本而不是压缩后的。
- 生成部分的KV Cache，老老实实全部保留。

在prefilling对应的KV Cache中保留了：
- 最近tokens (Observation Window)。
- 其余部分保留注意力分数高的tokens，以及他们的邻居们。



Content-Based: SnapKV

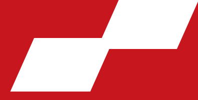


在LongBench上相较于H2O有更好表现:

	LLMs *	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code	
		NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P
LWMChat	All KV	**18.18**	**25.56**	40.94	24.57	19.39	10.49	**27.97**	24.9	**24.81**	71.0	60.9	39.73	3.17	3.5	44.4	43.82
	SnapKV: 1024	18.02	23.73	40.25	24.61	**19.84**	10.77	19.79	24.44	23.53	70.0	**61.42**	39.64	1.67	3.0	43.34	44.0
	SnapKV: 2048	17.92	25.03	**41.38**	24.49	19.38	**11.34**	21.6	24.22	24.36	70.0	61.11	39.91	2.17	4.0	44.46	**44.92**
	SnapKV: 4096	17.92	25.47	40.76	**24.92**	19.53	11.27	25.34	**25.42**	24.58	70.5	61.08	39.62	**3.17**	**4.0**	**44.49**	44.08
	H2O: 4096	13.17	24.82	20.01	16.86	9.74	7.2	25.77	23.26	23.83	**71.0**	61.06	**40.33**	0.0	0.0	41.52	40.97
LongChat	All KV	**20.88**	**29.36**	**43.2**	33.05	24.58	**14.66**	**30.89**	22.76	**26.61**	**66.5**	**83.99**	**40.83**	0.0	30.5	54.89	**59.05**
	SnapKV: 1024	19.32	26.6	37.93	34.15	23.34	12.71	23.45	21.81	24.93	65.0	80.88	38.19	0.0	31.0	53.63	57.62
	SnapKV: 2048	19.28	28.81	40.26	**35.31**	23.75	13.44	26.3	22.29	25.73	66.0	79.93	39.59	0.0	**31.0**	**56.05**	58.61
	SnapKV: 4096	20.68	29.34	42.21	33.95	**24.88**	14.15	28.55	**23.11**	26.45	66.0	**81.25**	**40.52**	0.0	29.5	54.79	58.81
	H2O: 4096	19.31	28.3	37.75	30.51	23.06	11.76	27.55	21.37	26.49	66.0	75.8	39.92	0.0	25.5	53.56	55.53
Mistral	All KV	**26.82**	33.06	49.28	**42.77**	27.33	19.27	**32.85**	24.25	27.06	71.0	86.23	42.98	2.75	86.98	55.51	**52.88**
	SnapKV: 1024	25.54	29.51	49.25	40.94	25.7	**19.42**	25.89	23.82	26.11	69.5	**86.48**	42.06	2.98	**88.56**	55.65	51.87
	SnapKV: 2048	25.89	32.47	48.6	41.71	27.31	18.69	28.81	**24.5**	26.6	70.0	86.27	42.47	3.09	87.43	**55.93**	52.01
	SnapKV: 4096	26.41	**33.36**	**49.81**	42.32	**27.93**	18.76	30.74	24.19	**27.08**	**71.0**	86.25	**43.01**	2.73	86.18	55.62	52.65
	H2O: 4096	22.61	29.06	47.22	36.54	20.6	16.25	30.0	23.8	26.75	70.5	86.16	42.97	**3.46**	86.38	53.72	51.1
Mixtral	All KV	26.81	**37.06**	51.55	47.77	32.46	**26.59**	**34.25**	**26.05**	27.91	76.0	90.57	46.98	5.5	100.0	**69.07**	69.65
	SnapKV: 1024	26.01	34.65	51.58	**48.23**	32.67	25.92	27.77	25.0	27.25	74.5	90.42	46.48	5.5	99.5	69.02	68.98
	SnapKV: 2048	**27.12**	36.9	51.91	47.46	33.23	26.27	30.19	25.84	27.8	**76.0**	90.24	46.31	5.5	100.0	68.72	**70.01**
	SnapKV: 4096	26.46	37.03	**52.62**	47.71	**33.35**	26.45	32.64	25.87	**27.94**	75.5	**90.71**	**47.14**	5.5	**100.0**	68.81	69.56
	H2O: 4096	20.45	32.09	48.02	34.76	25.69	16.5	29.76	23.53	26.84	74.5	90.24	47.1	**7.06**	99.42	64.91	63.52

SnapKV: LLM Knows What You are Looking for Before Generation, Li et al, 2024

Content-Based: SnapKV



用1k Cache能使LWM-Text-Chat-1M在140k文本长度内准确完成大海捞针任务。

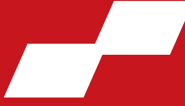
LWM-Text-Chat-1M with SnapKV

Depth Percent: 0.0, 11.0, 22.0, 33.0, 44.0, 56.0, 67.0, 78.0, 89.0, 100.0

Token Limit: 7513, 20538, 33564, 46590, 59615, 72641, 85667, 98692, 111718, 124744, 144231, 154462, 164692, 174923, 185154, 195385, 205615, 215846, 226077, 236308, 246538, 256769, 267000, 277231, 287462, 297692, 307923, 318154, 328385, 338615, 348846, 359077, 369308, 379538

Score: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

Content-Based: SnapKV



保留注意力分数高的token的邻居们很重要：保持要找的答案相关文本的整体性。

Mistral-7B-Instruct-v0.2 without Pooling

Depth Percent: 0.0, 11.0, 22.0, 33.0, 44.0, 56.0, 67.0, 78.0, 89.0, 100.0

Token Limit: 4000, 7000, 9000, 11000, 13000, 15000, 18000, 20000, 22000, 24000, 26000, 30000

Score: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

Mistral-7B-Instruct-v0.2 without Pooling

Depth Percent: 0.0, 11.0, 22.0, 33.0, 44.0, 56.0, 67.0, 78.0, 89.0, 100.0

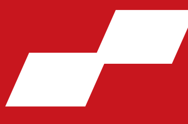
Token Limit: 5000, 7000, 9000, 11000, 13000, 16000, 18000, 20000, 22000, 24000, 27000, 30000

Score: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

SJTU

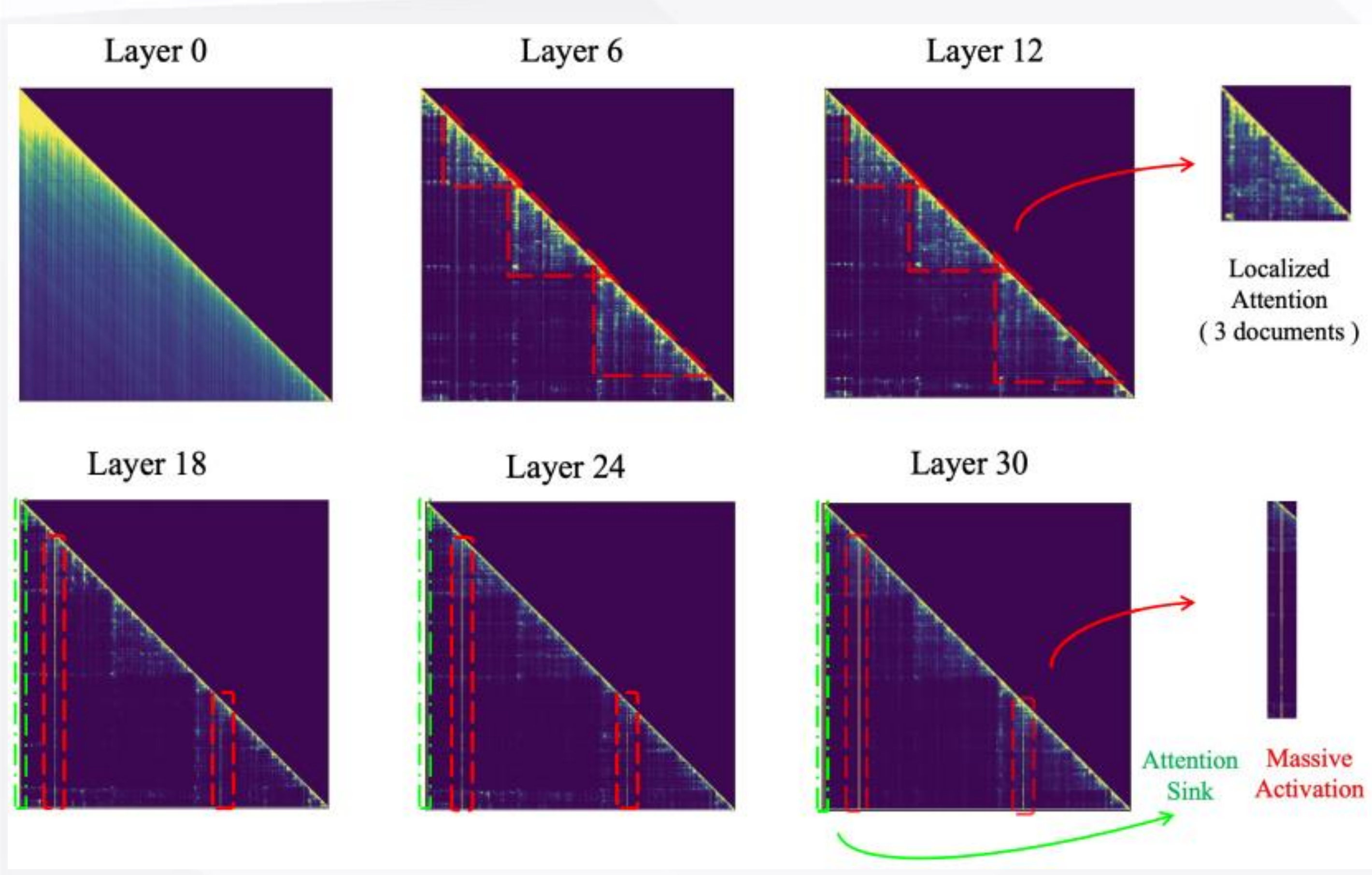
SnapKV: LLM Knows What You are Looking for Before Generation, Li et al, 2024

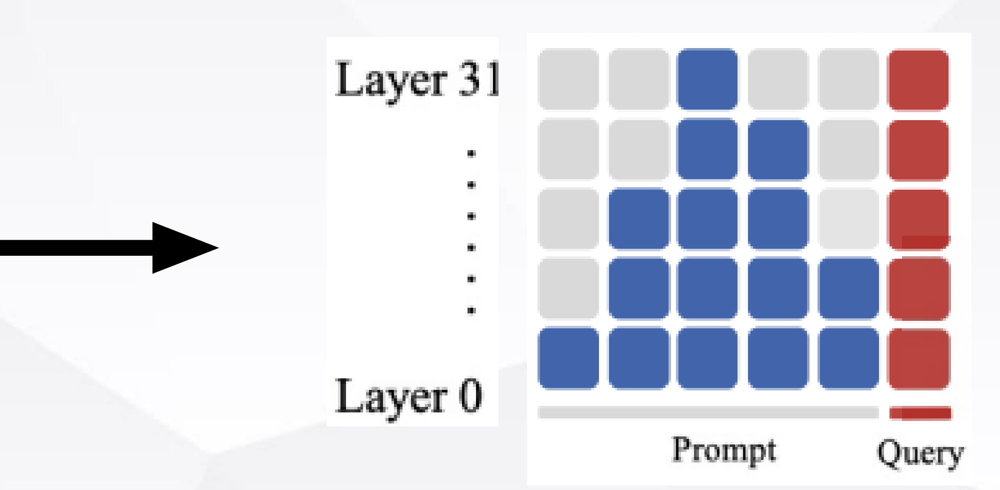
Content-Based: PyramidKV



发现：
不同层的attention maps有pyramidal patterns，随层数变深注意力由分散变集中。

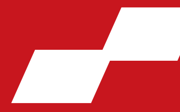
基于这个发现，PyramidKV改进了SnapKV
- Cache size浅层小，深层大
- 每层的压缩方式和SnapKV一致





PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling, Cai et al, 2024

Content-Based: PyramidKV

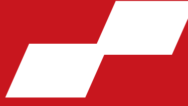


在LongBench上，模型LlaMa-3-8B-Instruct，Mistral-7B-Instruct，LlaMa-3-70B-Instruct均取得了比SnapKV,H2O,StreamingLLM更好的效果。

Method	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
	18409	3619	4559	9151	4887	11214	8734	10614	2113	5177	8209	6258	11141	9289	1235	4206	
							LlaMa-3-8B-Instruct, KV Size = Full										
FKV	25.70	29.75	41.12	45.55	35.87	22.35	25.63	23.03	26.21	73.00	90.56	41.88	4.67	69.25	58.05	50.77	41.46
							LlaMa-3-8B-Instruct, KV Size = 64										
SKV	19.86	9.09	27.89	**37.34**	28.35	**18.17**	15.86	20.80	16.41	38.50	85.92	36.32	5.22	69.00	51.78	48.38	33.05
H2O	20.80	11.34	27.03	37.25	30.01	17.94	18.29	21.49	19.13	38.00	84.70	**37.76**	**5.63**	69.33	**53.44**	**50.15**	33.89
SLM	17.44	8.68	22.25	35.37	**31.51**	15.97	15.46	20.06	14.64	38.00	72.33	29.10	5.42	**69.50**	46.14	45.09	30.43
Ours	**21.13**	**14.18**	**30.26**	35.12	23.76	16.17	**18.33**	**21.65**	**19.23**	**58.00**	**88.31**	37.07	5.23	**69.50**	52.61	45.74	**34.76**
							LlaMa-3-8B-Instruct, KV Size = 2048										
SKV	**25.86**	29.55	**41.10**	**44.99**	**35.80**	21.81	25.98	**23.40**	26.46	**73.50**	**90.56**	41.66	5.17	**69.25**	56.65	49.94	41.35
SLM	21.71	25.78	38.13	40.12	32.01	16.86	23.14	22.64	**26.48**	70.00	83.22	31.75	**5.74**	68.50	53.50	45.58	37.82
H2O	25.56	26.85	39.54	44.30	32.92	21.09	24.68	23.01	26.16	53.00	**90.56**	41.84	4.91	**69.25**	56.40	49.68	39.35
Ours	25.40	**29.71**	40.25	44.76	35.32	**21.98**	**26.83**	23.30	26.19	73.00	**90.56**	**42.14**	5.22	**69.25**	**58.76**	**51.18**	**41.49**

PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling, Cai et al, 2024

Content-Based: PyramidKV



在大海捞针实验上，PyramidKV相较于SnapKV也有提升。

LLaMA-3-8B - 8K Context Size

(a) FullKV, KV Cache Size = Full, Acc. = 100.0

(b) PyramidKV, KV Cache Size = 128, Acc. = 97.4

(c) SnapKV, KV Cache Size = 128, Acc. = 87.4

(d) H2O, KV Cache Size = 128, Acc. = 49.1

LLaMA-3-70B - 8K Context Size

(a) FullKV, KV Cache Size = Full, Acc. = 100.0

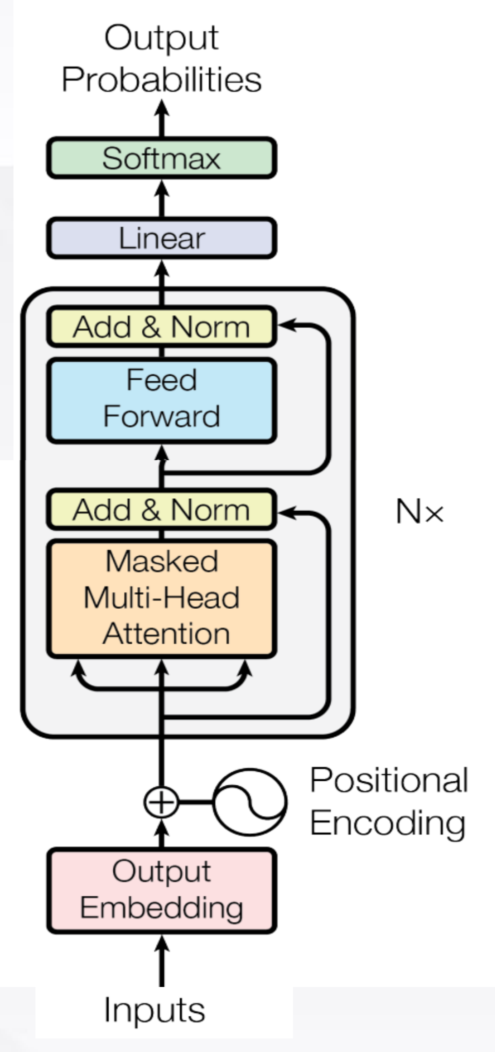
(b) PyramidKV, KV Cache Size = 128, Acc. = 100.0

(c) SnapKV, KV Cache Size = 128, Acc. = 98.6

(d) H2O, KV Cache Size = 128, Acc. = 82.3

PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling, Cai et al, 2024

目录



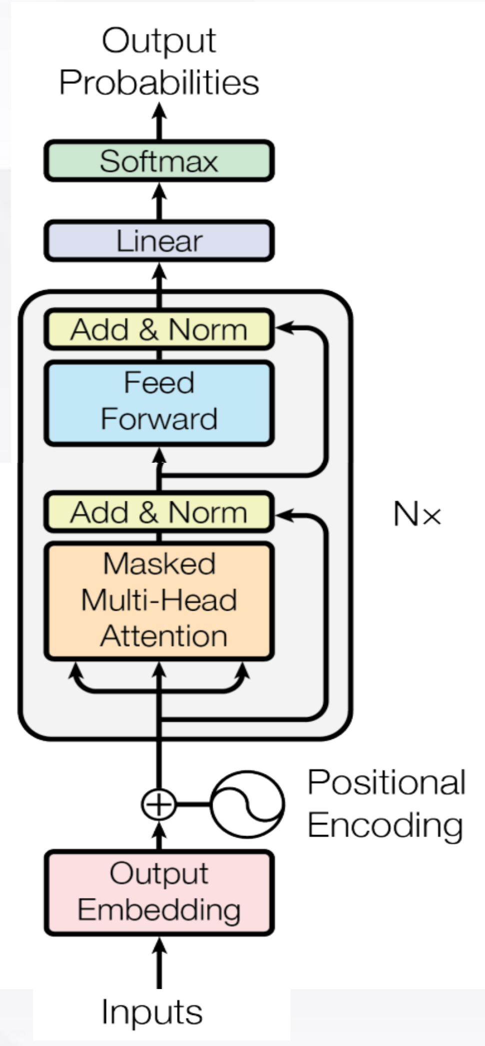
1 • Attention机制层面

2 • 逐层KV Cache压缩层面

3 • 层间KV Cache压缩层面

4 • 宏观计算架构适配层面

目录



3

- 层间KV Cache压缩层面

这一层面的方法，主要通过在不同层复用同一套 KV Cache，来降低显存开销

- Layer-Condensed KV
- Cross-Layer Attention
- You Only Cache Once
- Context Expansion with Parallel Encoding

SJTU

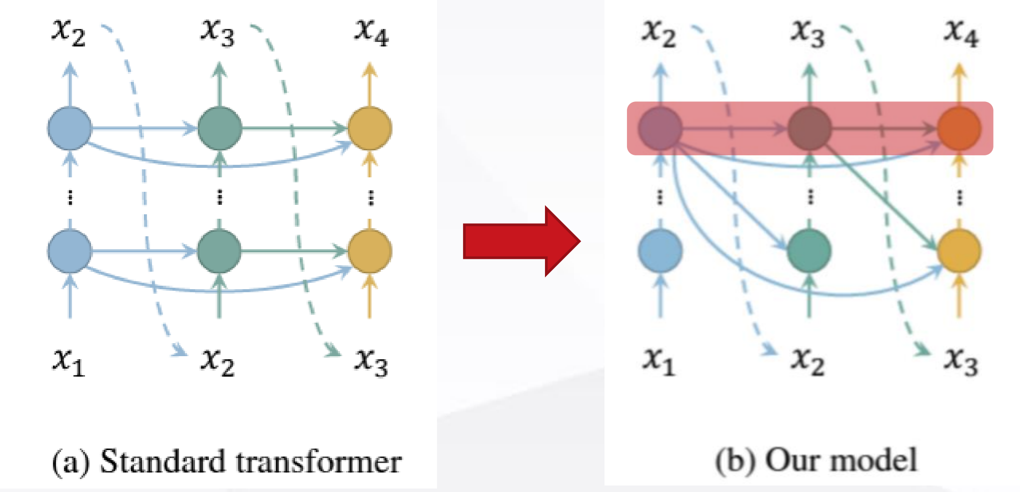
Layer-Condensed KV

Core Idea

顶层处的表示是最具信息量的，仅用顶层的KV cache，来减少KV Cache层的数量

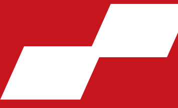
Challenge

- **性能下降**：在语言建模和下游任务中的性能有所下降，仅用同一层的 KVs 会破坏“在底层层关注语法信息，而在高层层关注语义信息”这种模式
- **顺序依赖**：每个token的计算依赖于前一个顶层的token，产生顺序依赖关系，破坏了并行训练



Haoyi Wu and Kewei Tu. Layer-Condensed KV Cache for Efficient Inference of Large Language Models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11175–11188. 2024.

Layer-Condensed KV

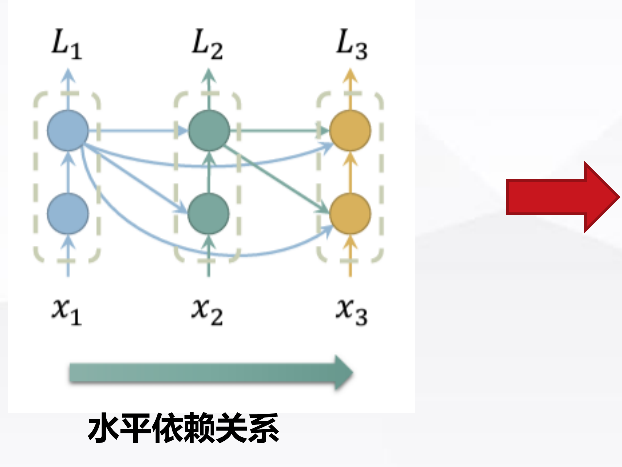


□ Challenge

1. **性能下降问题**：保留少量层的标准注意力，称为“预热层”，将LCKV应用于其余层；
2. **顺序依赖问题：** 设计一种新的并行训练方法；

a. 从顺序训练到并行训练

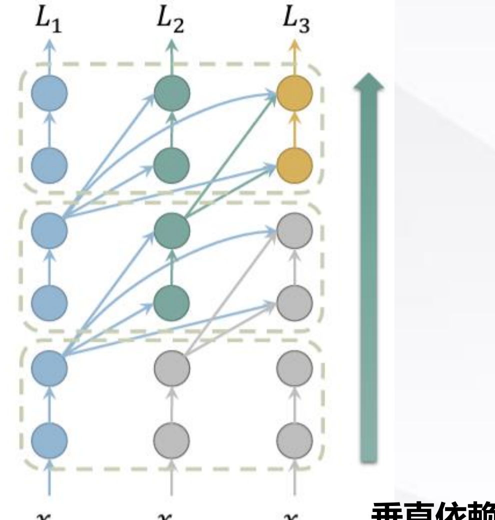
- 用垂直依赖关系替代了水平依赖关系，对所有token并行执行n次自下而上的attention；
- 但仍然需要**n**次迭代整个计算图；



水平依赖关系

b. 反向传播：梯度停止

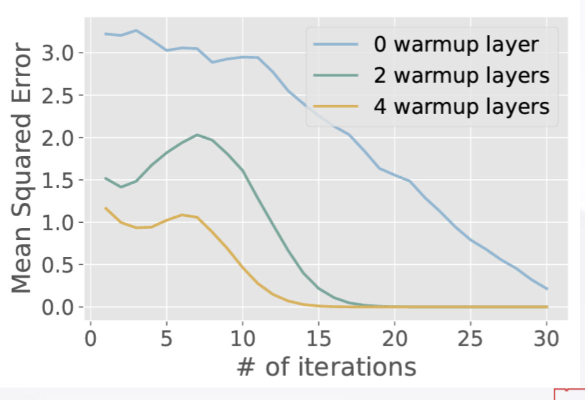
- 只在最后一次迭代计算loss;
- 遵循Transformer-xl，只将loss反向传播到最后b次迭代；



垂直依赖关系

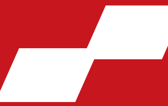
c. 前向传播：KV快速收敛

- 当n很大时，n-b次的前向过程代价仍然很大；
- 实验观察到，不需要老老实实算n-b次迭代；只需要少量迭代（7次左右），KV Cache就可以收敛；



Haoyi Wu and Kewei Tu. Layer-Condensed KV Cache for Efficient Inference of Large Language Models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11175–11188. 2024.

Layer-Condensed KV



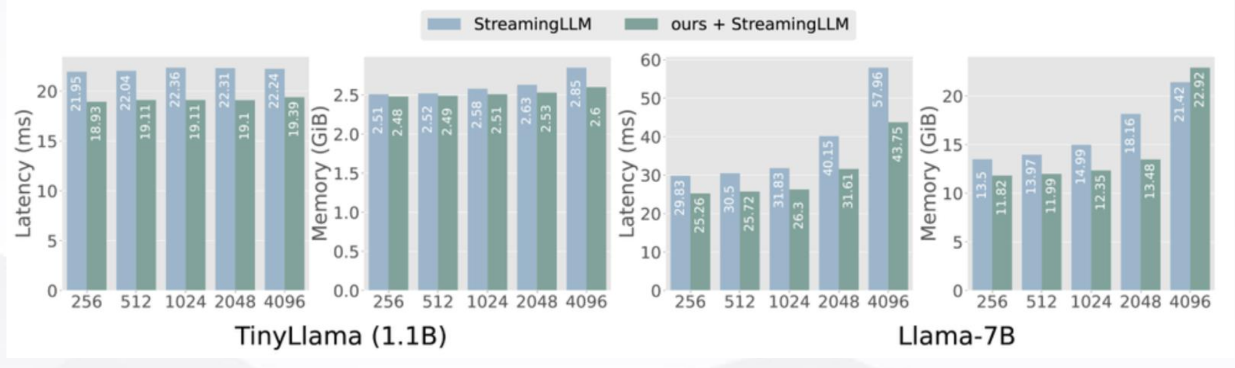
☐ Experiments

- **Generation Throughput**
  - 可以实现高达 26×倍的吞吐量提升

GPU	Model Size	Seq. Length	Batch Size: Llama	Batch Size: Ours $w=2$	Batch Size: Ours $w=10$	Throughput (tokens/s): Llama	Throughput (tokens/s): Ours $w=2$	Throughput (tokens/s): Ours $w=10$
RTX 3090	1.1B	5+8187	48	384 (8×)	119 (2.5×)	1424.96	4113.37 (2.9×)	2374.05 (1.7×)
		5+2043	239	1150 (4.8×)	289 (1.2×)	5142.86	10033.40 (2.0×)	7239.92 (1.4×)
	7B	5+8187	1	12 (12×)	4 (4×)	32.02	151.91 (4.7×)	83.80 (2.6×)
		2048+2048	2	23 (11.5×)	8 (4×)	56.98	171.65 (3.0×)	119.68 (2.1×)
		5+2043	5	64 (12.8×)	16 (3.2×)	140.88	534.02 (3.8×)	315.38 (2.2×)
		512+512	9	95 (10.6×)	32 (3.6×)	225.31	378.89 (1.7×)	380.60 (1.7×)
		512+1024	7	72 (10.3×)	16 (2.3×)	174.11	401.92 (2.3×)	310.05 (1.8×)
	30B (CPU-offload)	512+1024	4	83 (20.8×)	23 (5.8×)	0.23	5.99 (26.0×)	1.63 (7.1×)
A100	7B	2048+2048	15	128 (8.5×)	42 (2.8×)	141.10	421.02 (3.0×)	315.09 (2.2×)
	30B	2048+2048	1	32 (32×)	8 (8×)	14.10	108.29 (7.7×)	77.65 (5.5×)

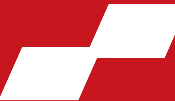
Table 1: Maximum generation batch size and throughput on an RTX 3090 (24GB) and an A100 (80GB) GPU respectively with different sequence lengths. Following Zhang et al. (2023), we use "$x+y$" to denote a prompt length of x and a generation length of y.

- **与StreamingLLM集成**
  - 实现了更低的延迟和内存消耗



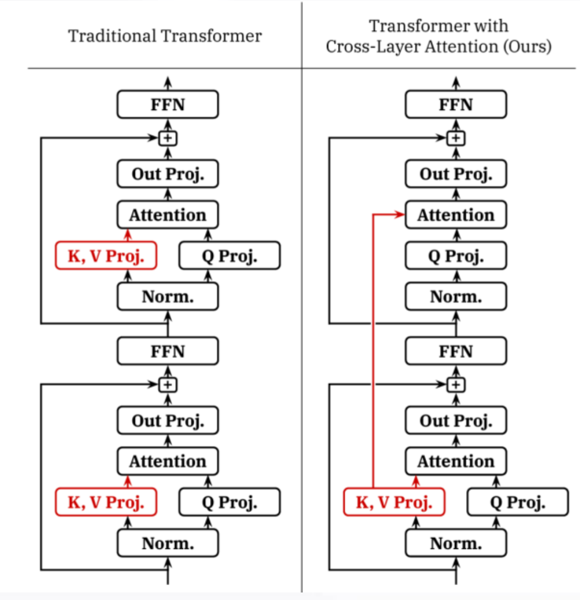
Haoyi Wu and Kewei Tu. Layer-Condensed KV Cache for Efficient Inference of Large Language Models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 11175–11188. 2024.

Cross-Layer Attention



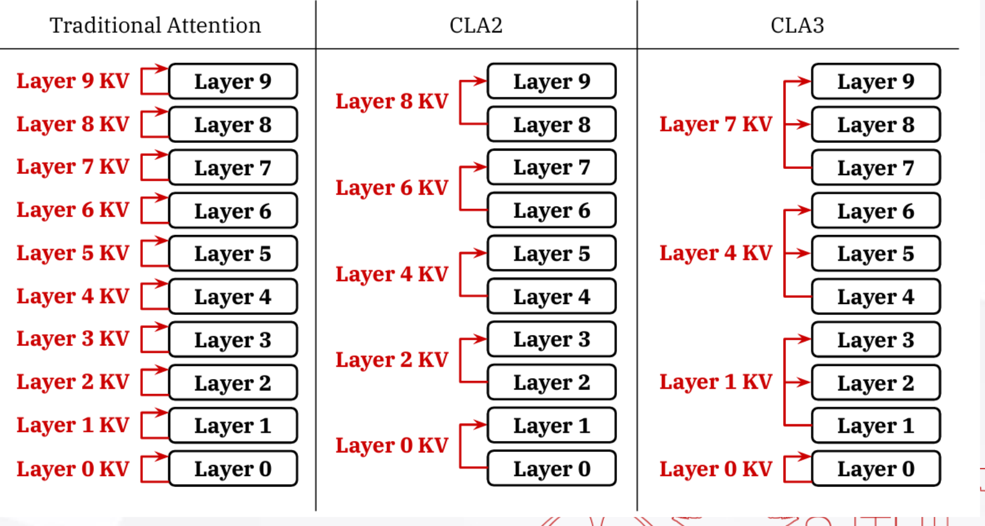
Core Idea

跨层注意力机制：在相邻层之间共享键和值；
- 只有小部分层计算键值对；
- 没有键值对的层，其注意力重用先前层的键值对。



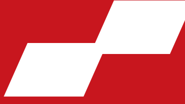
Strengths

1. 减少了训练期间实现的中间 KV 激活张量内存；
2. 实现不同的压缩率，缩小了等于共享因子的倍数；
3. CLA与MQA/GQA/MHA正交，可以与它们结合使用；



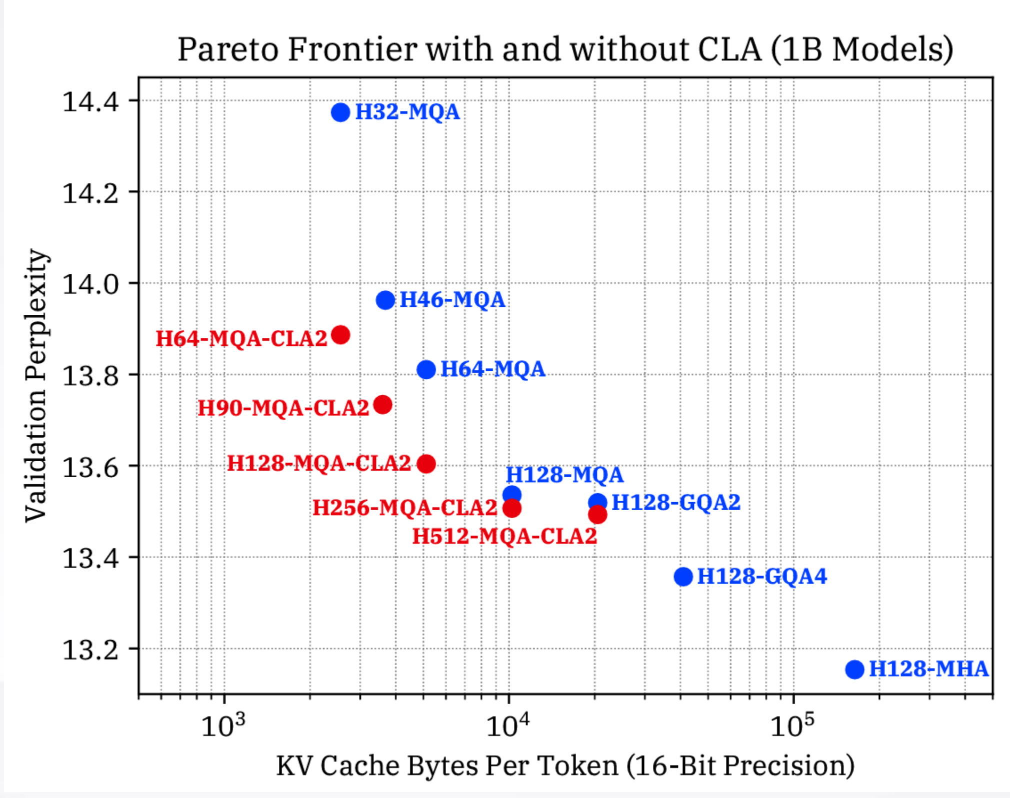
Brandon, William, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda and Jonathan Ragan-Kelley. “Reducing Transformer Key-Value Cache Size with Cross-Layer Attention.” Advances in Neural Information Processing Systems（2024）.

Cross-Layer Attention



Experiments

- CLA 相对于现有的多查询注意力 (MQA)，能够实现准确性/内存Pareto改进



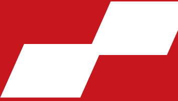
- 将 CLA 与 MQA/GQA相结合可以实现与普通 MQA 基线相比 2× 减少 KV Cache大小，而困惑度下降很小

Model	d_{head}	Query Heads	KV Heads	KV Layers	KV Bytes Per Token (16-Bit)	Validation Perplexity
Non-CLA Baselines						
H128-MHA	128	16	16	20	163 840	13.15
H128-GQA4	128	16	4	20	40 960	13.36
H128-GQA2	128	16	2	20	20 480	13.52
H128-MQA	128	16	1	20	10 240	13.54
H64-MQA	64	32	1	20	5120	13.81
H46-MQA	46	45	1	20	3680	13.96
H32-MQA	32	64	1	20	2560	14.37
MQA + CLA2 Models						
H512-MQA-CLA2	512	4	1	10	20 480	13.49
H256-MQA-CLA2	256	8	1	10	10 240	13.51
H128-MQA-CLA2	128	16	1	10	5120	13.60
H90-MQA-CLA2	90	22	1	10	3600	13.73
H64-MQA-CLA2	64	32	1	10	2560	13.89
GQA + CLA2 Models						
H256-GQA4-CLA2	256	8	4	10	40 960	13.38
H128-GQA4-CLA2	128	16	4	10	20 480	13.48
H128-GQA2-CLA2	128	16	2	10	10 240	13.59
MQA + CLA > 2 Models						
H128-MQA-CLA3	128	16	1	7	3584	13.77
H128-MQA-CLA4	128	16	1	5	2560	13.95
MQA + CLA2, Non-Uniform Sharing						
H128-MQA-CLA2-KeepEnds	128	16	1	11	5632	13.62
H128-MQA-CLA2-DenseFront	128	16	1	11	5632	13.75
H128-MQA-CLA2-DenseBack	128	16	1	11	5632	14.03

Table 1: Results of our 1B-scale design space exploration.

Brandon, William, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda and Jonathan Ragan-Kelley. “Reducing Transformer Key-Value Cache Size with Cross-Layer Attention. Advances in Neural Information Processing Systems（2024）.

You Only Cache Once



□Core Idea

解码器-解码器架构，只需要缓存一次KV pair

1. self-decoder

利用Efficient Self-Attention来生成全局 KV Cache

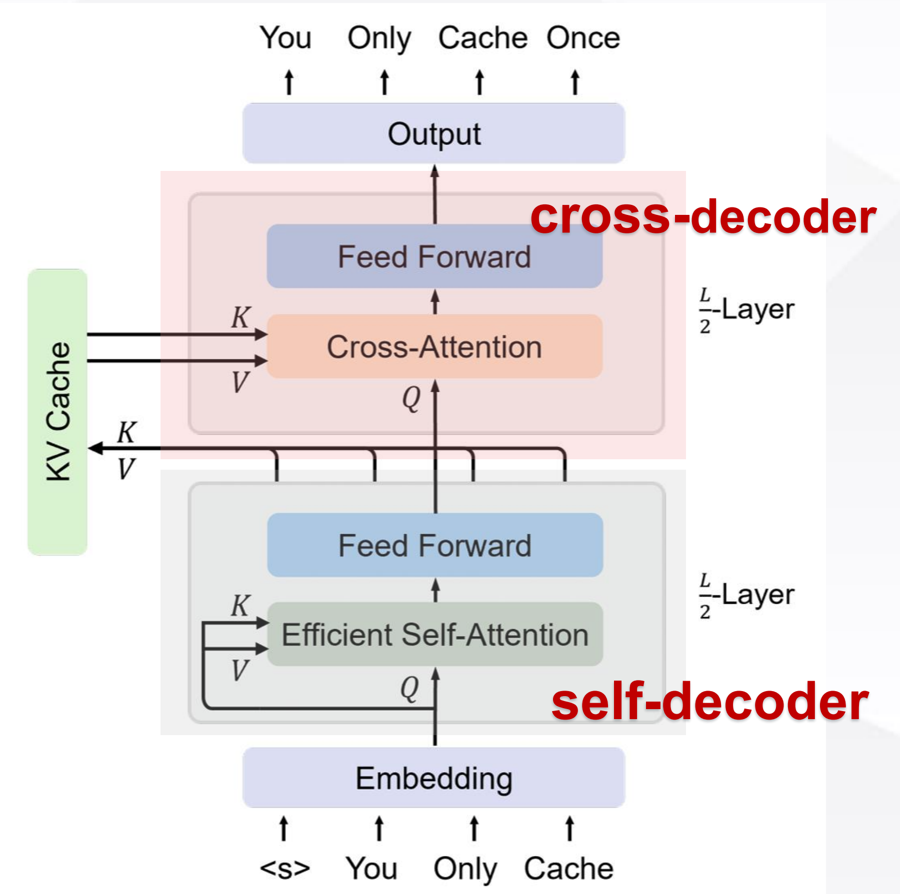
$$X^l = \mathrm{Self-Decoder}(X^{l-1}), l \in [1, \frac{L}{2}],$$

2. cross-decoder

使用交叉注意力机制来复用self-decoder生成的 KV Cache

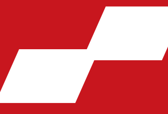
$$\hat{K} = \mathrm{LN}(X^{L/2})W_K, \quad \hat{V} = \mathrm{LN}(X^{L/2})W_V$$

$$X^l = \mathrm{Cross-Decoder}(X^{l-1}, \hat{K}, \hat{V}), l \in [\frac{L}{2}+1, L]$$



Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., & Wei, F. (2024). You Only Cache Once: Decoder-Decoder Architectures for Language Models. Advances in Neural Information Processing Systems（2024）.

You Only Cache Once



□Strengths

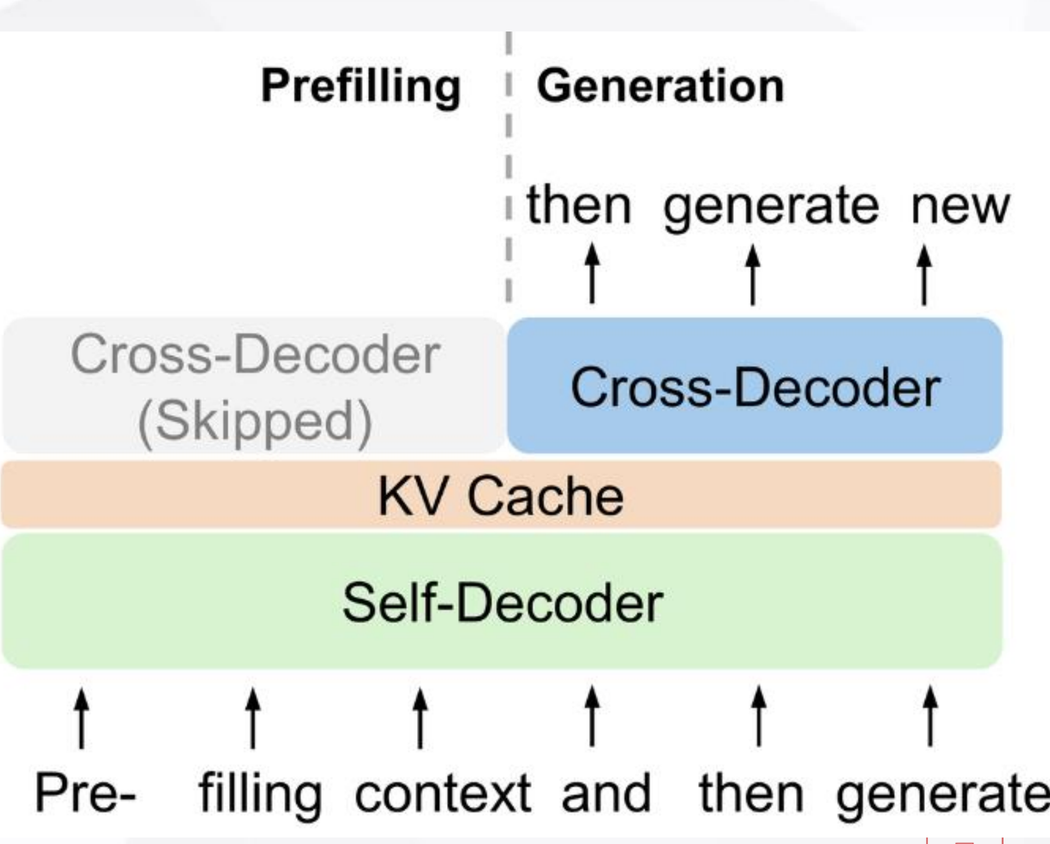
- **节省GPU内存并处理更多token**
  - Cache 数量为$O(N+CL)$，对于长序列，CL比N小得多，因此只需要$O(N)$，只需要缓存一次；
  - 在推理过程中，传统的Transformer必须存储 NL 个KV，YOCO节省了L倍的GPU内存，可以处理更多的token；
- **减少预填充（prefilling）时间并提高吞吐量**：
  - 在预填充阶段，forward process在进入Cross-Decoder之前即可提前退出（只需要计算一半的层）；
  - 有效自注意力模块比传统的自注意力模块更快；

	KV Cache Memory
Transformer	$\mathcal{O}(LND)$
YOCO	$\mathcal{O}((N+L)D)$

Table 1: Inference memory complexity of KV caches. N, L, D are the sequence length, number of layers, and hidden dimension.

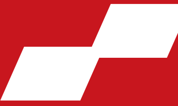
	Prefilling Time
Transformer	$\mathcal{O}(LN^2D)$
YOCO	$\mathcal{O}(LND)$

Table 2: Prefilling time complexity of attention modules. N, L, D are the same as above.



Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., & Wei, F. (2024). You Only Cache Once: Decoder-Decoder Architectures for Language Models. Advances in Neural Information Processing Systems（2024）.

You Only Cache Once



◻ Experiments

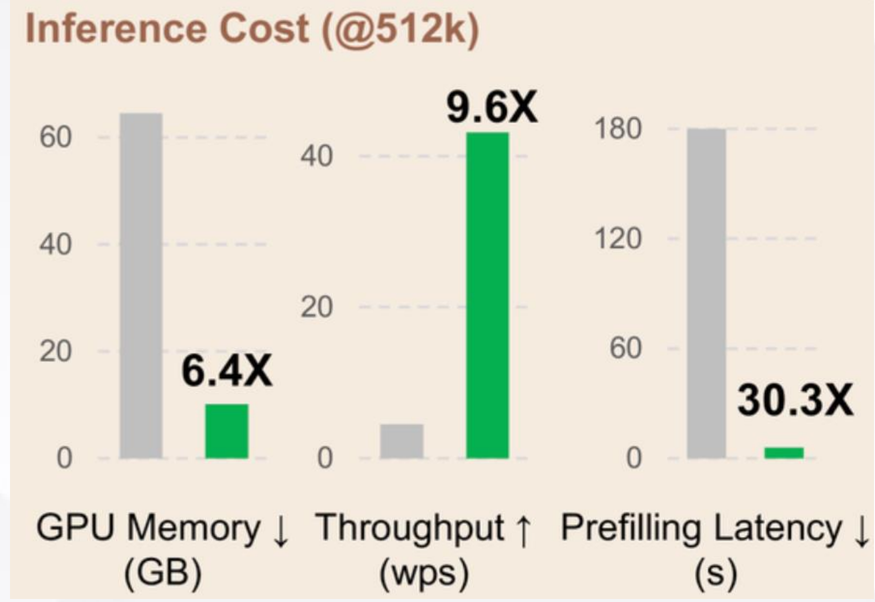
- 语言建模评估

扩展 Token训练，扩展到1M上下文长度

Model	ARC-C	ARC-E	BoolQ	Hellaswag	OBQA	PIQA	Winogrande	SciQ	Avg
Training with 1T tokens									
OpenLLaMA-3B-v2	0.339	0.676	**0.657**	**0.700**	0.260	0.767	0.629	**0.924**	0.619
StableLM-base-alpha-3B-v2	0.324	0.673	0.646	0.686	0.264	0.760	0.621	0.921	0.612
StableLM-3B-4E1T	—	0.666	—	—	—	**0.768**	0.632	0.914	—
YOCO-3B	**0.379**	**0.731**	0.645	0.689	**0.298**	0.763	**0.639**	**0.924**	**0.634**
Training with 1.6T tokens									
StableLM-3B-4E1T	—	0.688	—	—	—	0.762	0.627	0.913	—
YOCO-3B	0.396	0.733	0.644	0.698	0.300	0.764	0.631	0.921	0.636
Extending context length to 1M tokens									
YOCO-3B-1M	0.413	0.747	0.638	0.705	0.300	0.773	0.651	0.932	0.645

Table 3: Eval Harness [GTA+23] results compared with previous well-trained Transformer language models [TBMR, Tow, GL23]. We scale the 3B model to 1.6 trillion training tokens. The 1T and 1.6T results of StableLM-3B-4E1T are taken from its technical report [TBMR]. YOCO-3B-1M is extended to the context length of 1M tokens.

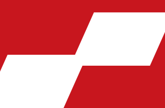
- 提高了推理内存、预填充延迟和吞吐量



SJTU

Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., & Wei, F. (2024). You Only Cache Once: Decoder-Decoder Architectures for Language Models. Advances in Neural Information Processing Systems（2024）.

Context Expansion with Parallel Encoding



Core Idea

两个组件，适用于任何decoder-only LM

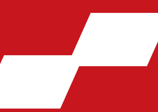
- **小型编码器**：采用小型编码器来逐块处理长输入；
- **交叉注意力块**：使冻结的解码器能够通过额外的交叉注意力模块来利用额外的上下文。

Strengths

- **长度泛化**：CEPE 不受位置编码约束的限制，因为长上下文被分段编码，每个有自己的位置编码。
- **效率**：标准的decoder模型需要O((m+n)Ld_{dec})，CEPE仅需要O(md_{enc}+nLd_{dec})；
- **降低训练成本**：只调整编码器和交叉注意力，保持LLM 冻结；

Howard Yen, Tianyu Gao, and Danqi Chen. Long-Context Language Modeling with Parallel Context Encoding. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2588–2610, 2024.

Context Expansion with Parallel Encoding



Experiments

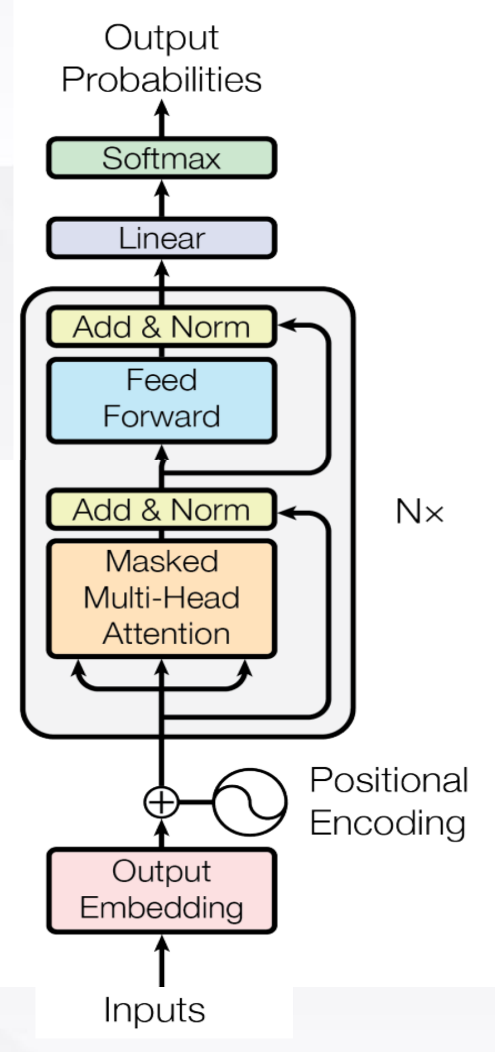
- 将 LLaMA-2 的上下文窗口扩展到 128K Token
- 仅用 1/6 的内存即可提供 10× 的吞吐量
- 在长上下文和检索增强任务上实现了高效训练和检索性能的平衡

	ArXiv	Book	PG19	ProofPile	CodeParrot	Throughput	Mem. (GB)
Total Tokens = 4,096							
LLaMA-2	2.597	**6.282**	7.614	2.409	**1.735**	1.00×	**19.2**
LLaMA-2-32K	2.601	6.621	7.945	2.414	1.785	1.00×	**19.2**
YaRN-64K	2.651	6.337	**7.326**	2.457	1.764	1.04×	**19.2**
CEPE	**2.579**	6.292	7.536	**2.396**	1.763	**1.31**×	19.8
Total Tokens = 8,192							
LLaMA-2	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	-	-
LLaMA-2-32K	2.505	6.339	7.744	2.221	1.729	1.00×	24.9
YaRN-64K	2.561	6.077	**7.146**	2.267	**1.714**	2.52×	24.8
RePlug	2.589	6.149	7.554	2.307	1.728	0.17×	**18.8**
StreamingLLM	2.740	6.327	7.783	2.437	1.806	1.94×	20.0
CEPE	**2.496**	**6.049**	7.372	**2.219**	1.715	**3.48**×	22.6
Total Tokens = 32,768							
LLaMA-2-32K	**2.322**	6.178	7.420	**2.158**	1.664	1.00×	59.1
YaRN-64K	2.359	**5.884**	**6.809**	2.193	**1.640**	1.03×	58.9
StreamingLLM	2.752	6.358	7.627	2.503	1.853	1.16×	**20.0**
CEPE	2.421	6.015	7.204	2.218	1.702	**3.72**×	25.6
Total Tokens = 131,072							
LLaMA-2-32K	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	-	-
YaRN-64K	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	$> 10^3$	-	-
YaRN-128K	2.359	5.270	6.306	2.242	**1.264**	1.00×	235.6
StreamingLLM	2.371	5.058	6.681	2.270	1.280	2.56×	**20.0**
CEPE	**2.217**	**4.869**	**6.305**	**2.099**	1.266	**9.90**×	38.6

	ArXiv	Book	C4-RP	CC	Github	StackEx	Wiki	Avg.
$k = 0$ ($T = 2,048$)								
LLaMA-2	**3.541**	**6.524**	**6.916**	**5.564**	**1.865**	**4.043**	**4.816**	**4.753**
LLaMA-2-32K	3.561	6.892	7.798	5.931	1.932	4.262	4.958	5.048
YaRN-64K	3.633	6.631	7.164	5.701	1.930	4.164	4.837	4.866
$k = 8$ ($T = 4,096$)								
LLaMA-2	3.602	6.581	6.963	5.348	1.829	4.044	4.815	4.740
LLaMA-2-32K	3.642	6.985	7.767	5.645	1.893	4.270	4.988	5.027
YaRN-64K	3.752	6.718	7.218	5.466	1.894	4.178	4.847	4.868
RePlug	3.535	6.494	6.895	5.395	1.833	4.029	4.798	4.711
CEPE	**3.486**	**6.481**	**6.884**	**5.319**	**1.793**	**3.709**	**4.302**	**4.568**
$k = 20$ ($T = 7,168$)								
RePlug	3.531	6.490	6.894	5.386	1.830	4.028	4.795	4.708
CEPE	**3.475**	**6.463**	**6.875**	**5.266**	**1.782**	**3.703**	**4.296**	**4.551**
$k = 50$ ($T = 14,848$)								
RePlug	3.530	6.491	6.899	5.392	1.830	4.028	4.794	4.709
CEPE	**3.467**	**6.457**	**6.881**	**5.273**	**1.777**	**3.701**	**4.292**	**4.550**

Howard Yen, Tianyu Gao, and Danqi Chen. Long-Context Language Modeling with Parallel Context Encoding. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2588–2610, 2024.

目录



1 • Attention机制层面

2 • 逐层KV Cache压缩层面

3 • 层间KV Cache压缩层面

4 • 宏观计算架构适配层面

SJTU

目录

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Masked
Multi-Head
Attention
Nx
Positional
Encoding
Output
Embedding
Inputs

4

• 宏观计算架构适配层面

这一层面的方法并不改变模型本身，而是从模型所运行的计算环境的特点出发寻求解决方案

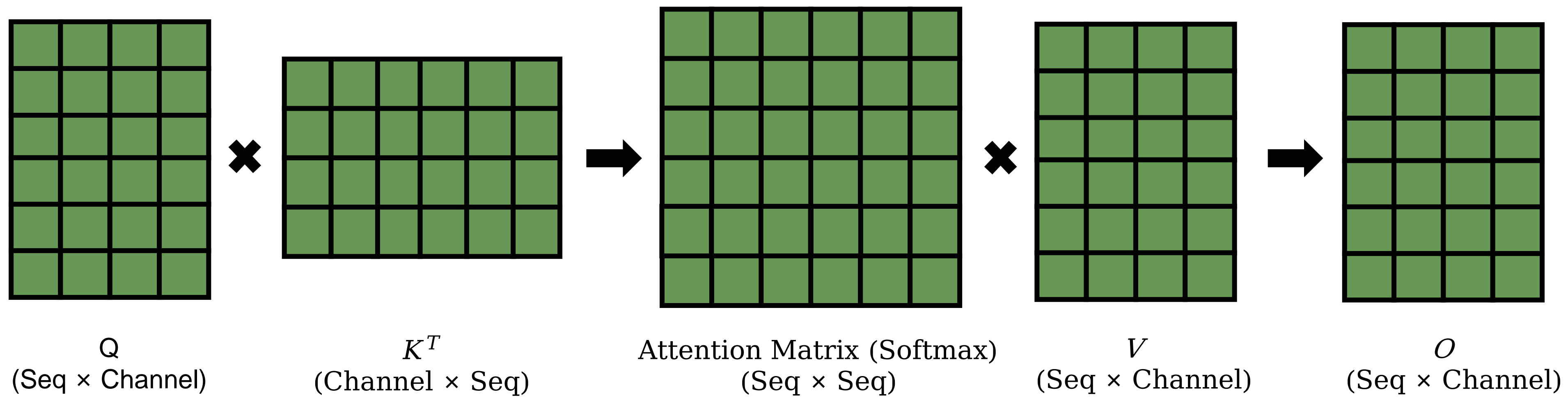
- Flash Attention系列
- Flash Decoding系列
- Speculative Decoding

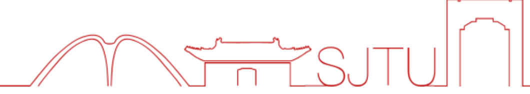
SJTU

Motivation

Original Attention

- Quadratic **complexity in sequence length**, which is a challenge for long sequence
- Reading and writing the attention matrix to and from HBM

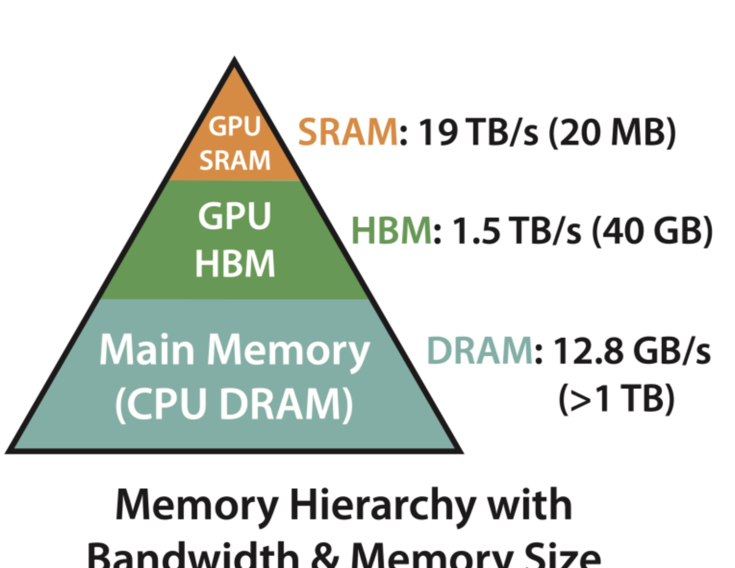




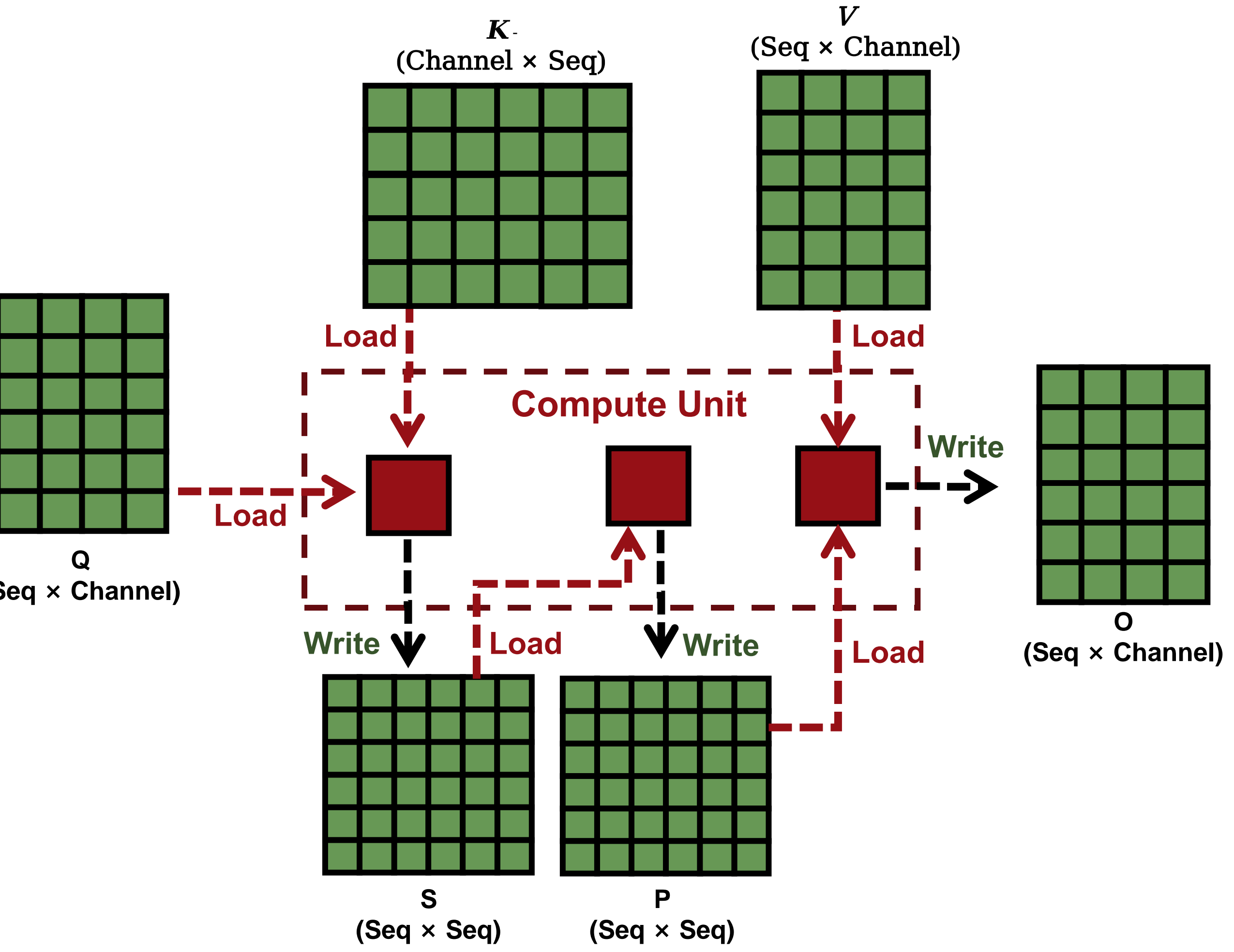
Flash Attention

Motivation

Original Attention VS FlashAttention



Memory Hierarchy with Bandwidth & Memory Size

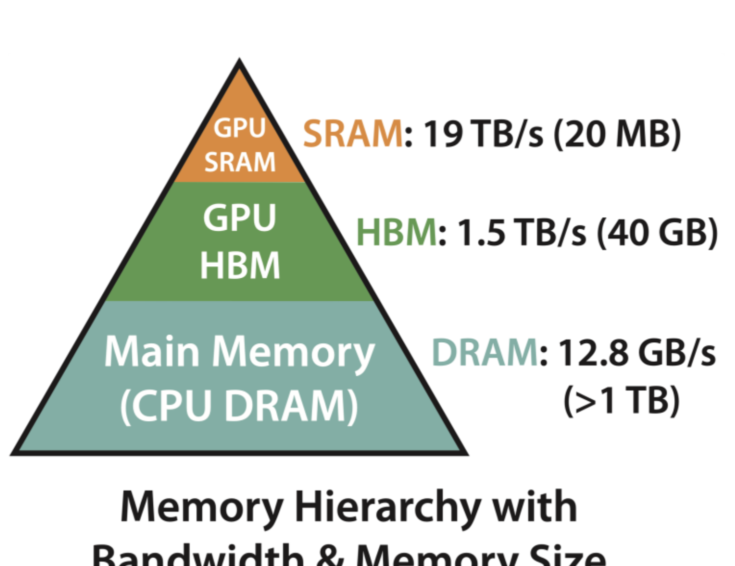


- Quadratic **complexity in sequence length**, which is a challenge for long sequence
- Reading and writing the attention matrix to and from HBM

Flash Attention

Motivation

Original Attention VS FlashAttention



Memory Hierarchy with Bandwidth & Memory Size

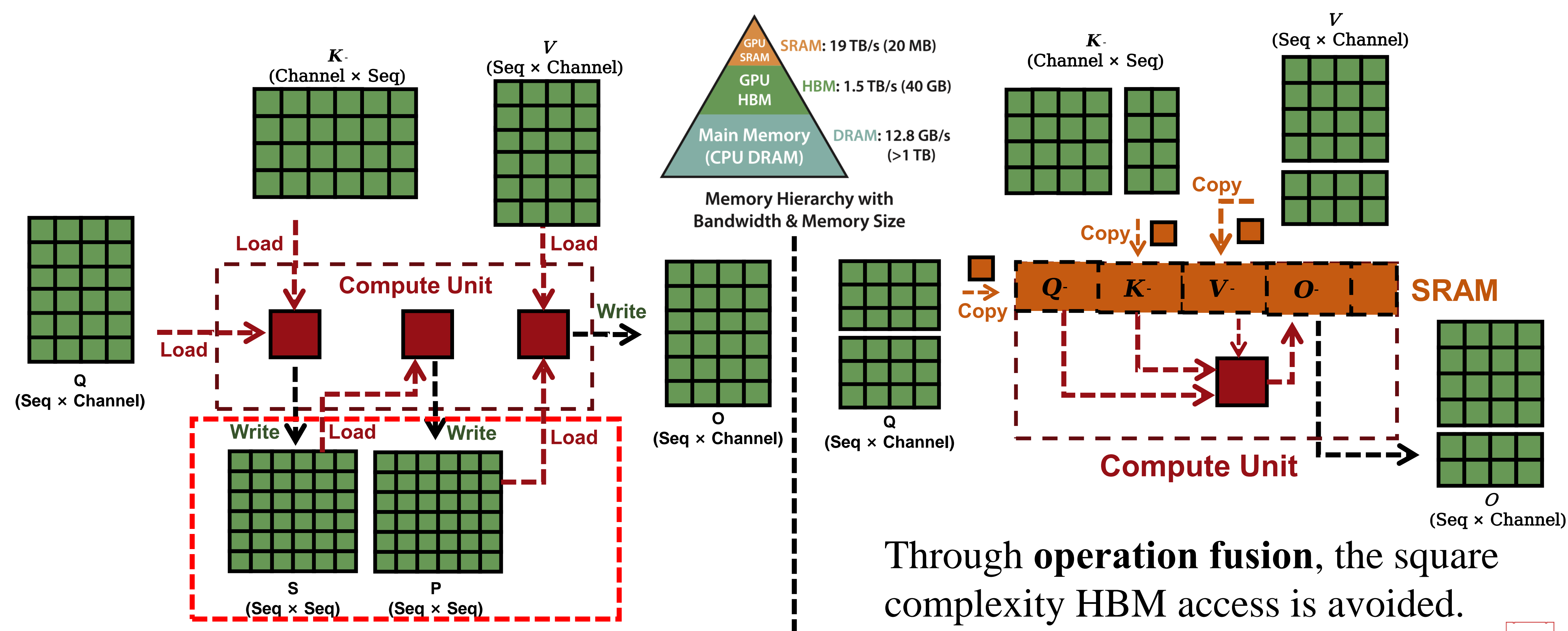
K
(Channel × Seq)
V
(Seq × Channel)
Load
Load
Compute Unit
Write
Load
Q
(Seq × Channel)
Write
Load
Write
Load
O
(Seq × Channel)
S
(Seq × Seq)
P
(Seq × Seq)

Quadratic Conplexity Overhead !

- Quadratic **conplexity in sequence length**, which is a challenge for long sequence
- Reading and writing the attention matrix to and from HBM

SJTU

Flash Attention

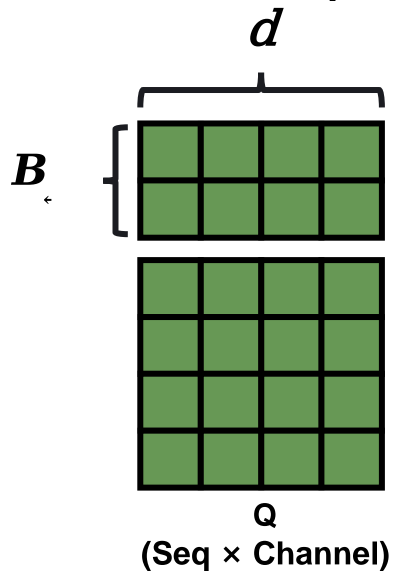


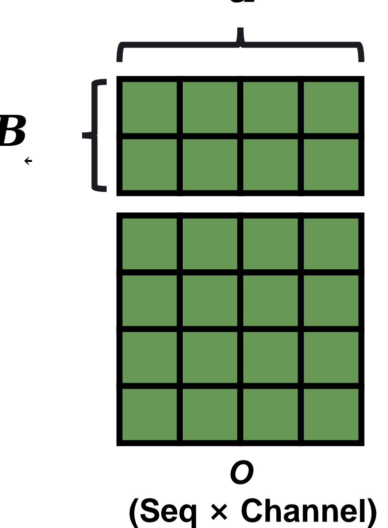
Through **operation fusion**, the square complexity HBM access is avoided.

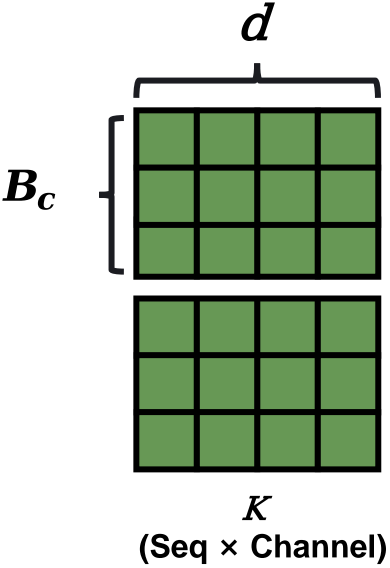
Method

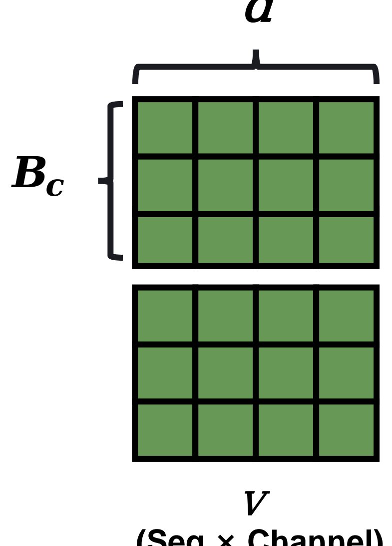
Step 1: Divide into blocks

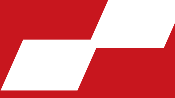
- Matrics Q, K, V, O in HBM
- Split Q, O into N / B_r , K, V into N / B_c block, where $B_c = \left\lceil \frac{M}{md} \right\rceil$, $B_r = \mathrm{Min}(\left\lceil \frac{M}{md} \right\rceil, d)$











Method

Step 2: On chip compute

Flash Attention

Method

Step 2: On chip compute

Outer Loop j

K
(Channel × Seq)

V
(Seq × Channel)

Copy

Copy

Inner Loop i

Inner Loop i

Copy

Write

Q
(Seq × Channel)

Q
$(B \times d)$

K
$(d \times B_c)$

S
$(B \times B_c)$
not materialize

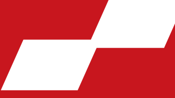
V
$(B_c \times d)$

O
$(B \times d)$

O
(Seq × Channel)

Compute Unit

Flash Attention



Method

Step 2: On chip compute

Outer Loop j

K (Channel × Seq)

V (Seq × Channel)

Copy

Copy

Inner Loop i

Copy

Q (Seq × Channel)

Q $(B_r \times d)$ × K $(d \times B_c)$ → S $(B_r \times B_c)$ not materialize × V $(B_c \times d)$ → O $(B_r \times d)$

Write

Inner Loop i

O (Seq × Channel)

Compute Unit

SJTU

Method

Step 2: On chip compute

Method

Step 2: On chip compute

Outer Loop j

K (Channel × Seq)

V (Seq × Channel)

Copy

Copy

Inner Loop i

Copy

Write

Inner Loop i

Q (Seq × Channel)

Q_i $(B_r \times d)$

K_j^T $(d \times B_c)$

S_{ij} $(B_r \times B_c)$ not materialize

V_j $(B_c \times d)$

O_i $(B_r \times d)$

O (Seq × Channel)

Compute Unit

Method

Step 2: On chip compute

Outer Loop j

K
(Channel × Seq)

V
(Seq × Channel)

Copy

Copy

Inner Loop i

Q
(Seq × Channel)

Copy

Q
(*B* × *d*)

K
(*d* × B_c)

S
(*B* × B_c)
not materialize

V
(B_c × *d*)

O
(*B* × *d*)

Write

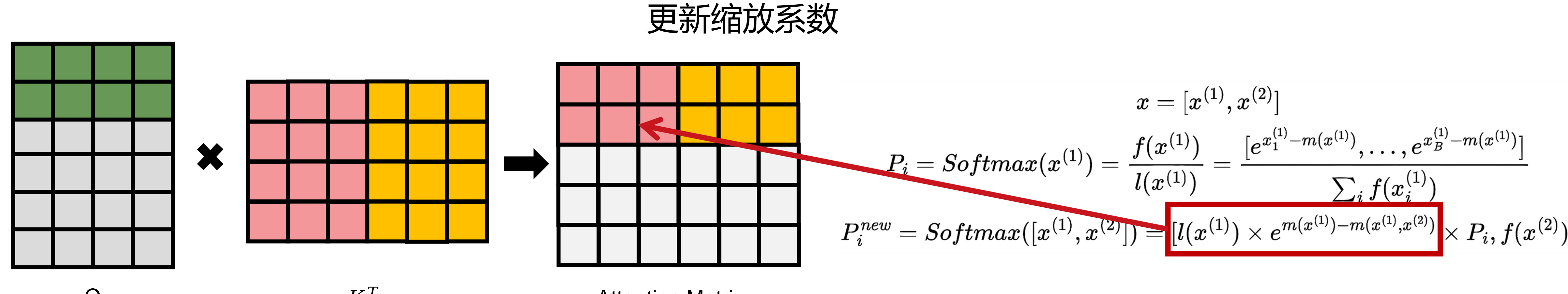
Inner Loop i

O
(Seq × Channel)

Compute Unit

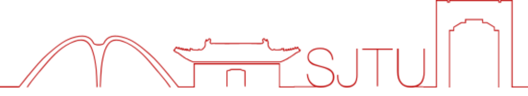
Dynamic Updating

- Quadratic **complexity in sequence length**, which is a challenge for long sequence
- Reading and writing the attention matrix to and from HBM



Backward Recomputation

- Standard attention requires $S, P \in \mathbb{R}^{N\times N}$, where S is attention score matrix and P is Softmax(S)
- FlashAttention store output O and Softmax normalization factor (m, l), which can be used to **reconstruct S, P in SRAM**



Flash Attention

Avoid matrix S and P HBM access by calculating on-chip

IO Complexity

- Standard attention requires $\Theta(Nd+N^2)$ HBM access
- FlashAttention requires $\Theta(N^2d^2/M)$

SJTU

Flash Attention V2

- Q blocks are independent
- Switch inner and outer loop
- Block-level parallelism

switch

Inner Loop j

Outer Loop i

Inner Loop i

K
(Channel × Seq)

V
(Seq × Channel)

Copy

Copy

Copy

Copy

Copy

Q
(Seq × Channel)

Q K V O

Write

Compute Unit

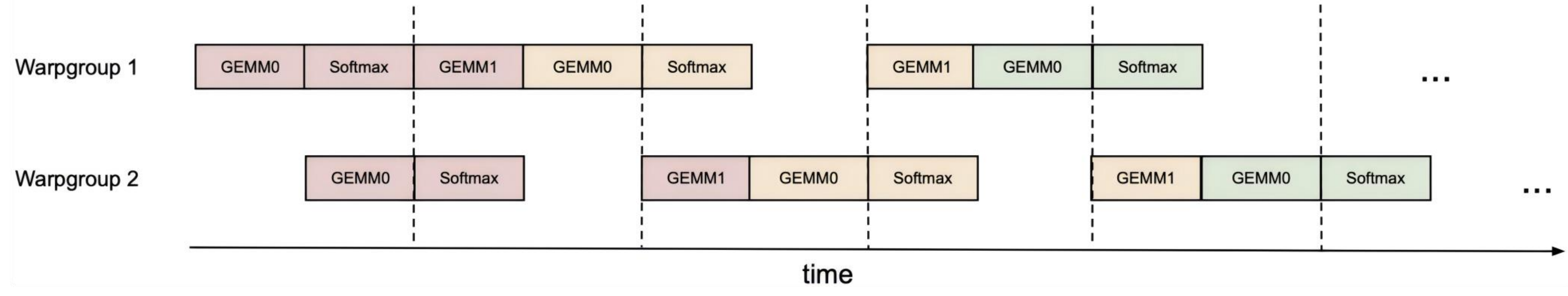
O
(Seq × Channel)

Block-level Parallelism

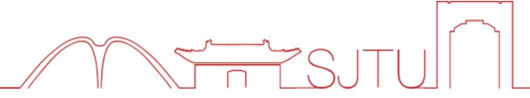
SJTU

Flash Attention V3

- TMA(Hopper) performs asynchronous data transfers in a DMA-like manner, allowing for instruction parallelism



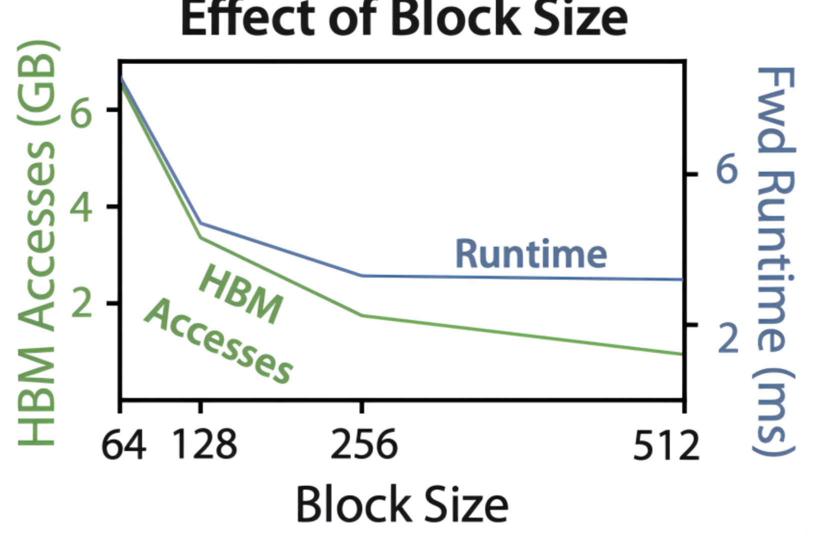
- Support FP8 quantization



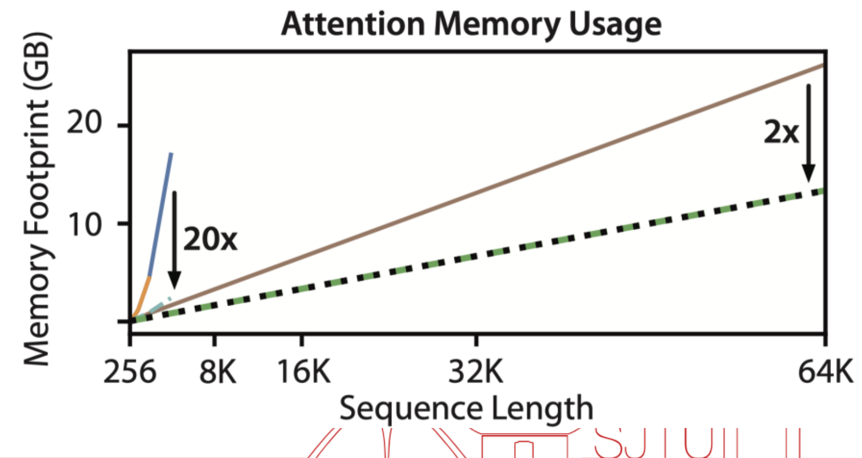
Flash Attention系列

Evaluation

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	35.3	4.4
Runtime (ms)	35.1	11.7

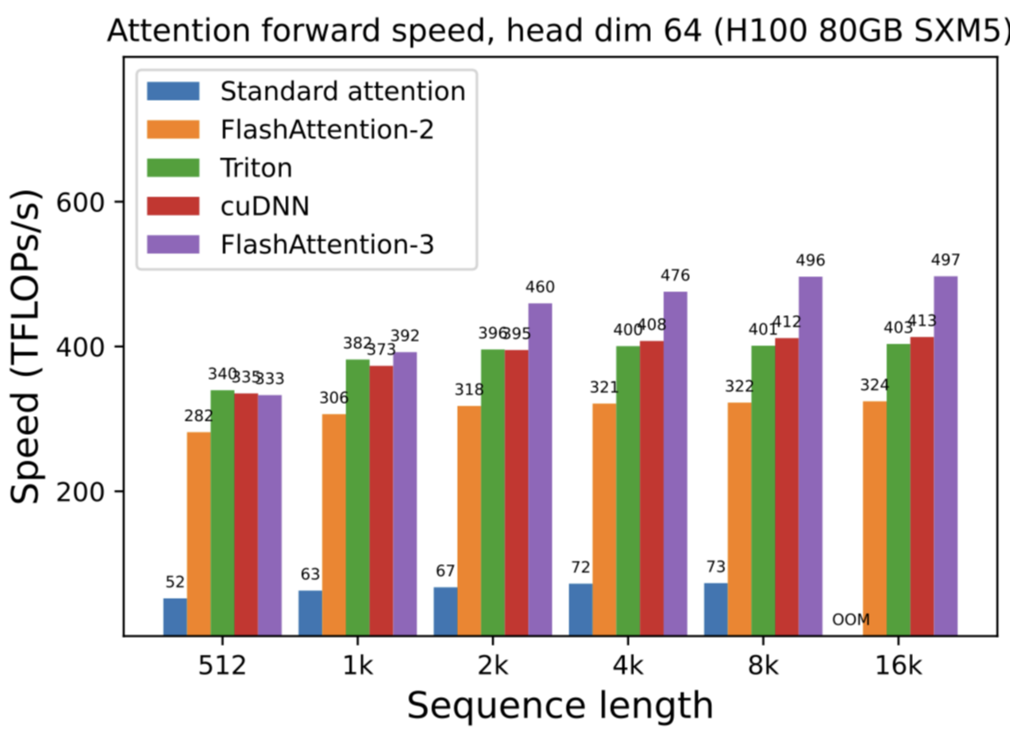


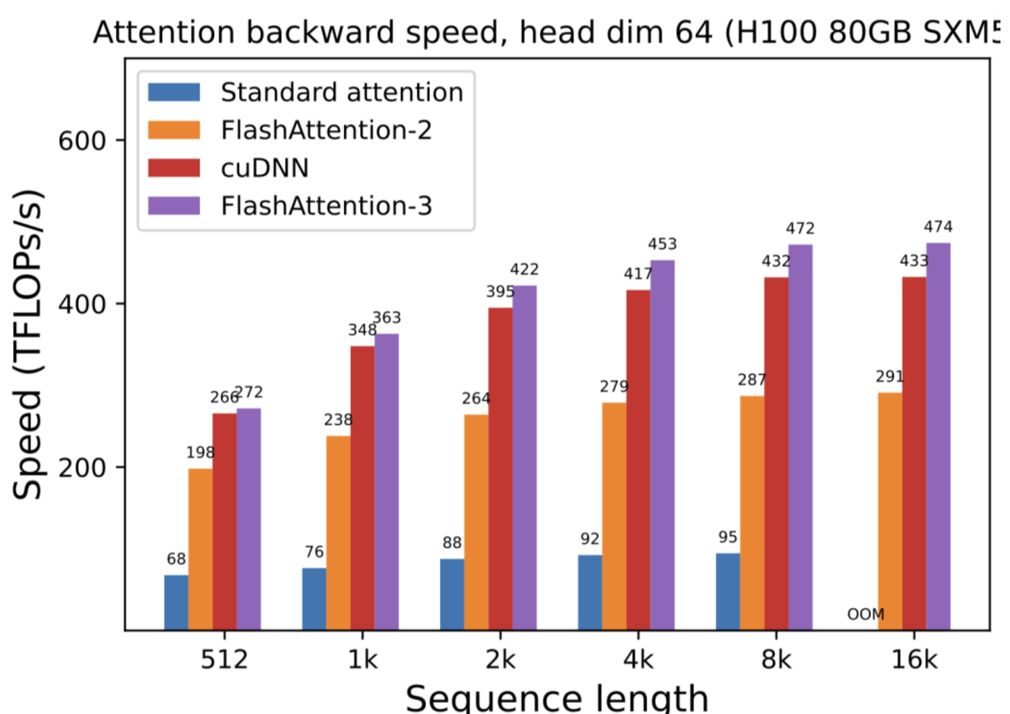
- FlashAttention reduces HBM access (upleft)
- Fewer HBM access result in faster end2end runtime (upright)
- FlashAttention reduces memory footprint (right)

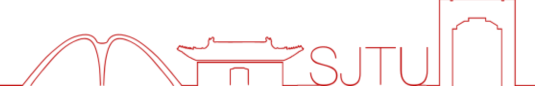


Flash Attention系列

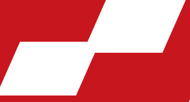
Evaluation





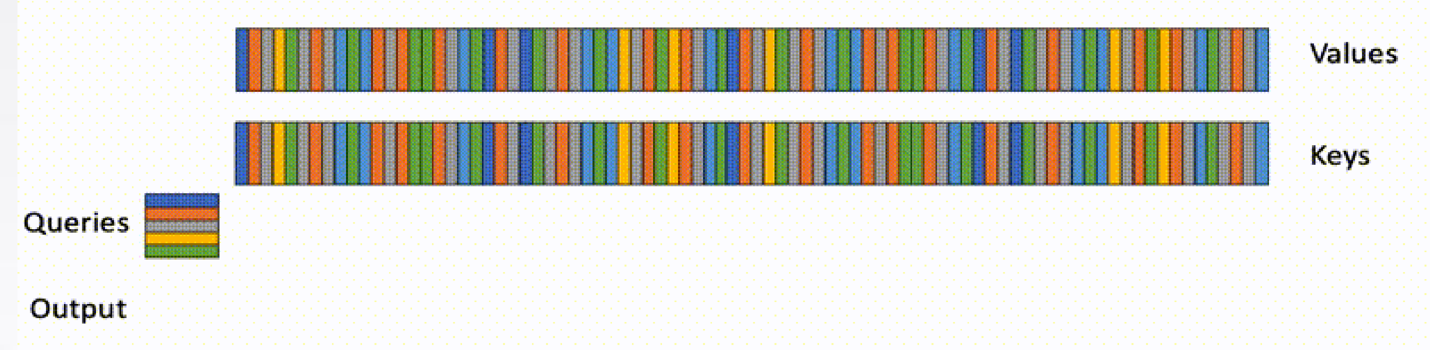


Flash Decoding



Flash Attention：

并行对象为Query与batch size，优化主要聚焦于Train阶段

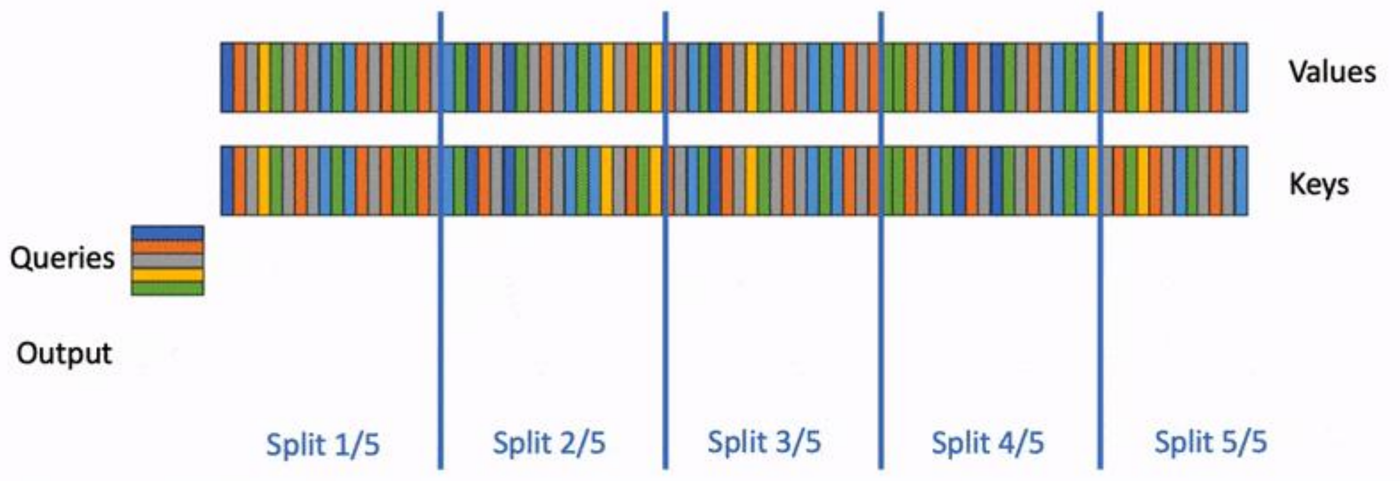


LLM Inference：

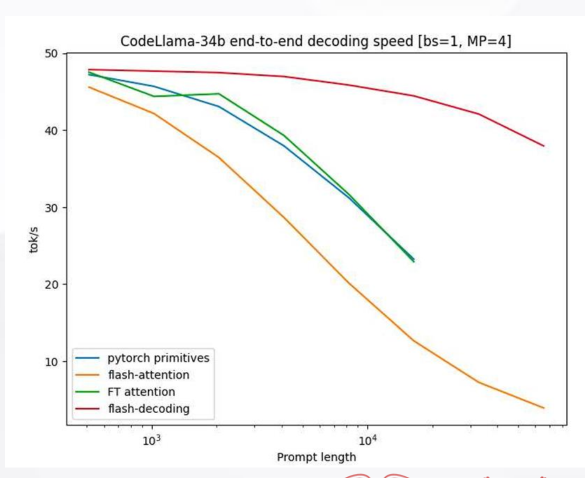
Query 长度为1，对GPU利用率低

能否在Key/Values上进行并行?

Flash Decoding



- **将key/value划分为多个chunk**
- **每个chunk内部进行Flash Attention**，对于chunk中的每一行，会额外记录该特征维度的attention的log-sum-exp
- 利用log-sum-exp，**chunk间执行reduction**得到唯一的输出。



FlashDecoding++

Flash-Decoding很快，但还**不够快**：Chunk间synchronization的时间高达18.8%

$softmax(x)$

x_1 x_2 x_3 x_4 ⋯ x_{d-1} x_d

$m(x) = max(x_i)$

$f(x) = \left(e^{x_1 - m(x)}, e^{x_2 - m(x)}, \ldots, e^{x_d - m(x)}\right)$

$l(x) = \sum_i f_i(x)$

$softmax(x) = \frac{f(x)}{l(x)}$

High parallelism ×
Low memory ×
Synchronization-free √

(a) Original softmax

$softmax(x')$ $softmax(x'')$

x_1 x_2 x_3 x_4 ⋯ x_{d-1} x_d

Calculate $m(x'), f(x'), l(x'), softmax(x')$

Calculate $m(x''), f(x''), l(x''), softmax(x'')$

Update $softmax(x')$ with $m(x'), f(x'), l(x'), m(x''), f(x''), l(x'')$

Update $softmax(x'')$ with $m(x'), f(x'), l(x'), m(x''), f(x''), l(x'')$

Process next partial vector

High parallelism √
Low memory √
Synchronization-free ×

(b) Partial softmax

$softmax(x')$ $softmax(x'')$

x_1 x_2 x_3 x_4 ⋯ x_{d-1} x_d

$m(x') = m(x'') =$ a unified max value ϕ

Calculate $f(x')$, $l(x')$

Calculate $f(x'')$, $l(x'')$

……

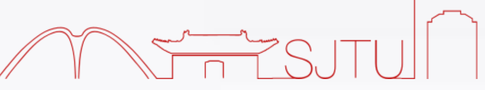
Calculate $softmax(x)$

High parallelism √
Low memory √
Synchronization-free √

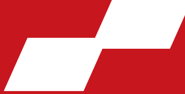
(c) Partial softmax with unified max value

Figure 4: Comparison of different softmax computation schemes. (a) Softmax computation for the whole vector. (b) Computing partial softmax for each partial vector, and a synchronized update operation is required for all partial softmax results. (c) Computing partial softmax using a unified max value, and each partial vector is processed individually without synchronized update.

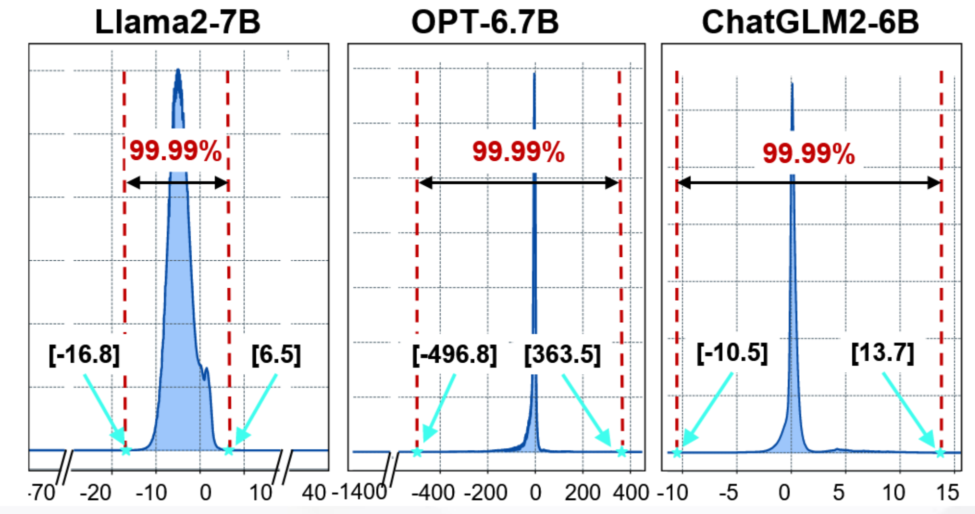
Partial softmax中的$\mathrm{m}(x)$能不能全局共用?



FlashDecoding++



- 部分模型的大部分x_i分布集中

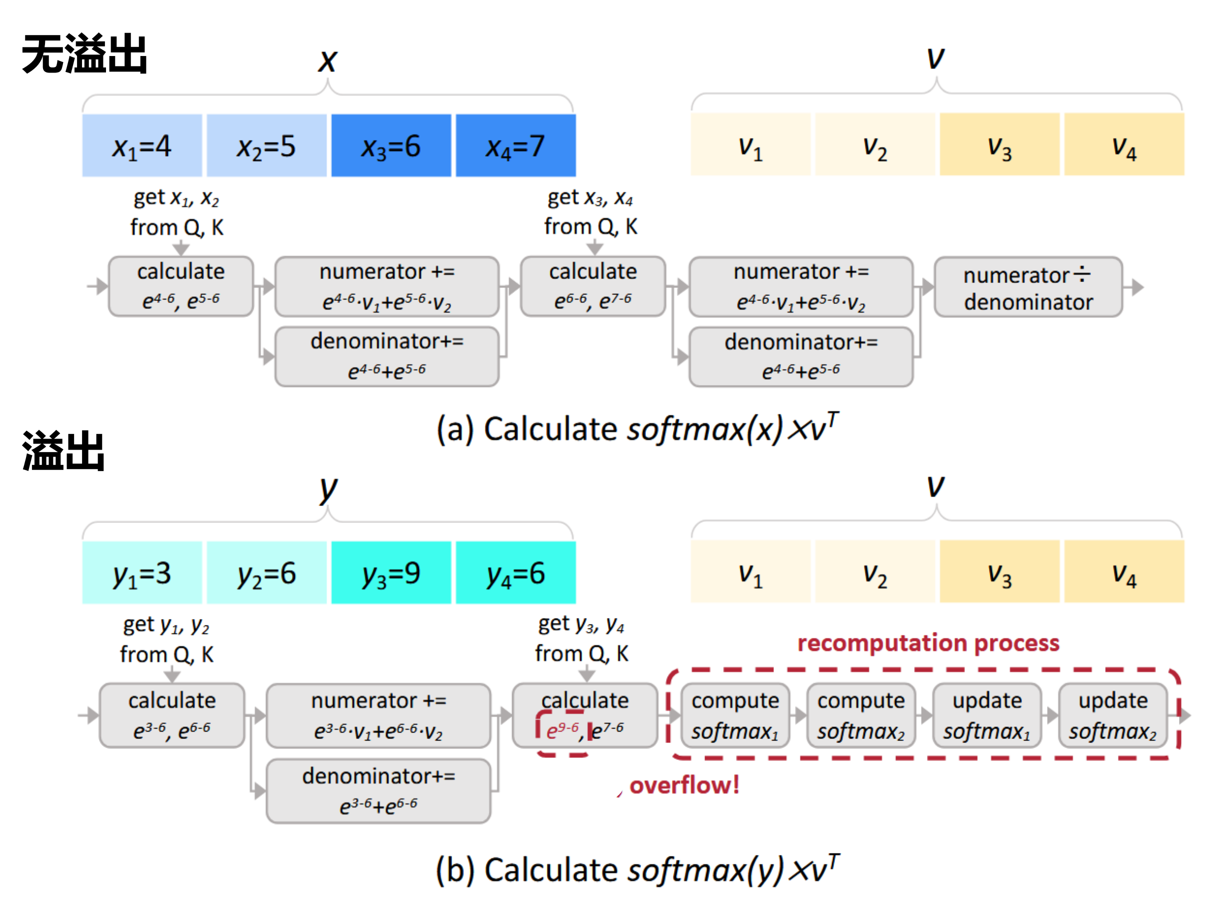


设定上下限a, b与 ϕ，作为unified $m(x)$

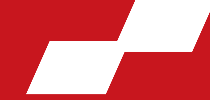
$$a < x_i - \phi < b$$

$$a = -3, b = 3$$

溢出时，退化回Flash-Decoding使用partial softmax



FlashDecoding++



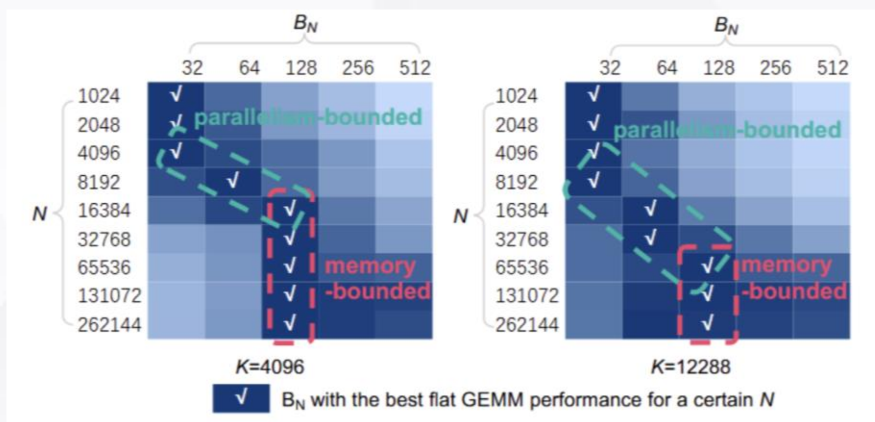
- **GEMV/Flat GEMM**对于GPU利用效率低：

  Inference阶段query为1，现有库对于query矩阵默认填充到64（Tensor Core 最低支持8位）

- **Memory-bounded**与**Parallelism-bounded**优化相冲突，**引入双buffer隐藏内存访问延时**

  假设Q，K矩阵分别为：$M \times K, K \times N$

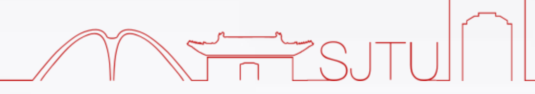
  B_N, B_K为分块大小



- **结合Tensor Core和CUDA核心的性能差异使用启发式算法选择硬件资源**

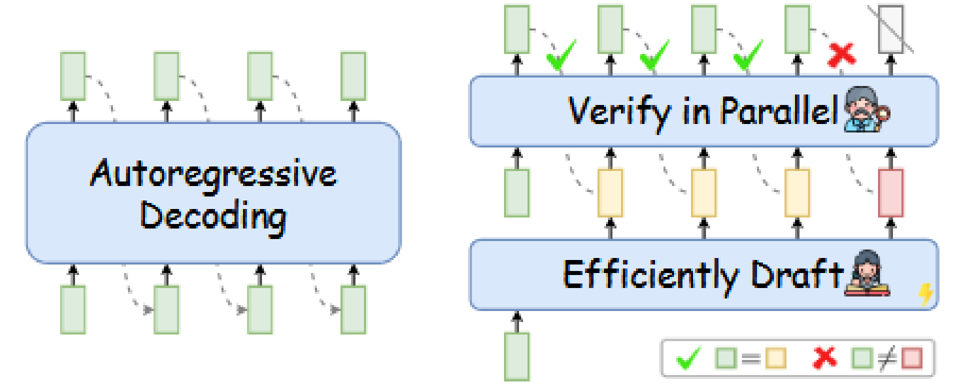
  Tensor Core（cuBLAS库）擅长于处理大型矩阵乘法

  CUDA（FastGEMV）在对小批量的GEMV计算效率更优



Speculative Decoding

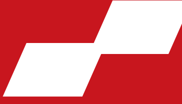
- 如何减少LLM在Inference时的Latency?
- Latency的来源：
  - 模型的forward计算？ ×
  - 大量的参数从HBM（High-Bandwidth Memory）移动到Cache √
- 简单的Token可以用更少的计算开销进行预测
- 思路：**减少参数的反复读写，提高Decode阶段的并行性**



SJTU

Xia, Heming, et al. "Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding." arXiv preprint arXiv:2401.07851 (2024).

Speculative Decoding



```
[START] japan ' s benchmark bond n
[START] japan ' s benchmark nikkei 22 ,5
[START] japan ' s benchmark nikkei 225 index rose 22 ,6
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 , points
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 0 1
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 , in
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in tokyo late
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

Draft: Drafted Token

Verify: Rejected Token
Corrected Token

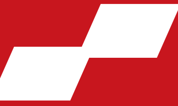
- **Draft-then-Verify:**

使用Draft Model M_P 加速Target Model M_q

Target Model M_q只需要验证Drafted Tokens

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.

Speculative Decoding

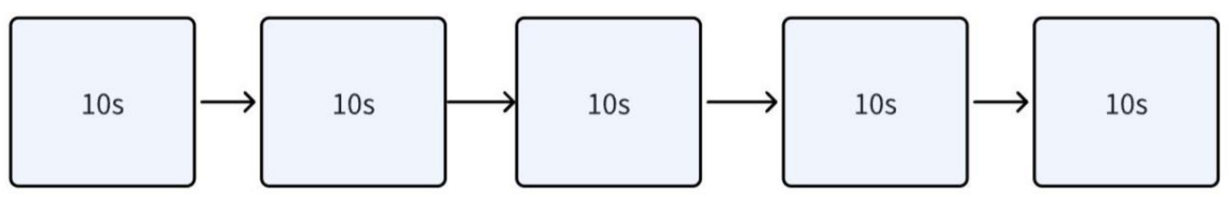


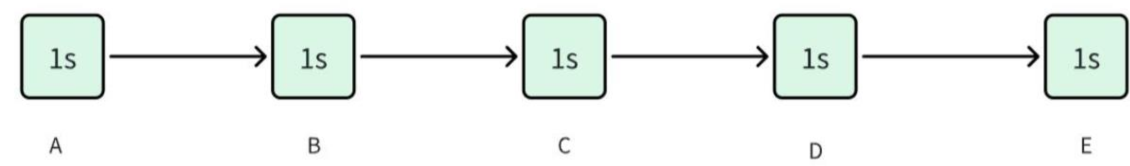
- **Draft（推测）：**

在decoding阶段，利用Draft Model M_P draft 多个token

$$p_1, \dots, p_K = DRAFT\left(x_{\leq t}, M_p\right)$$

$$\tilde{x} \sim p_i, i = 1, \dots, K$$





Speculative Decoding 使用一个系列中**较小的LM**作为Draft Model 为其中的大模型加速推理

Draft Model	Target Model
OPT-125M	OPT-70B
T5-small	T5-XXL
LAMDA (137B)	LAMDA (100M)

优点:

1. 不需要额外的训练资源
2. 同系列小模型对齐度更高

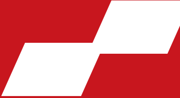
缺点:

1. 某些模型无同系列小模型，需要额外训练开销
2. 需要部署两个模型

SJTU

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.

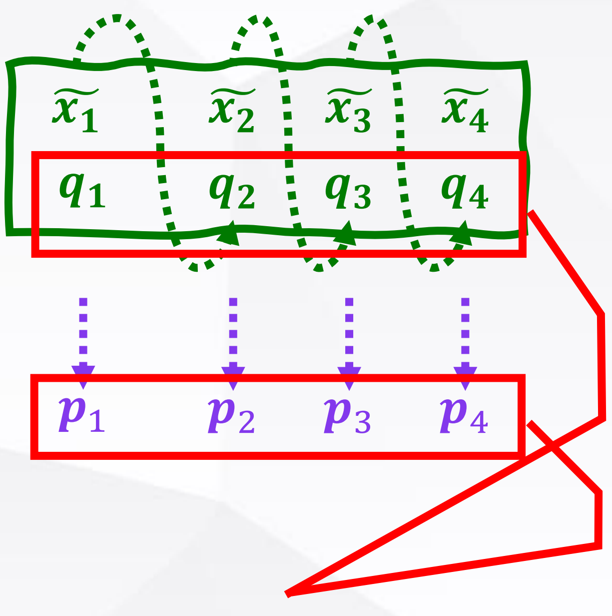
Speculative Decoding



[START] japan ' s benchmark nikkei 225 index rose 22 .6

x_1 x_2 x_3 … … … x_t

prompt



小模型逐字生成，并给出每个token的概率

大模型并行读入，一次forward生成每个token的概率

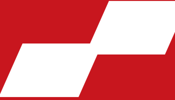
$$r < \min\left(1, \frac{p_i(\tilde{x}_i)}{q_i(\tilde{x}_i)}\right), r \sim U[0,1]$$

通过比较两个模型给出的概率值来判断是否要采纳小模型的生成结果

SJTU

[1]. Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.
[2]. Chen, Charlie, et al. "Accelerating large language model decoding with speculative sampling." arXiv preprint arXiv:2302.01318 (2023).

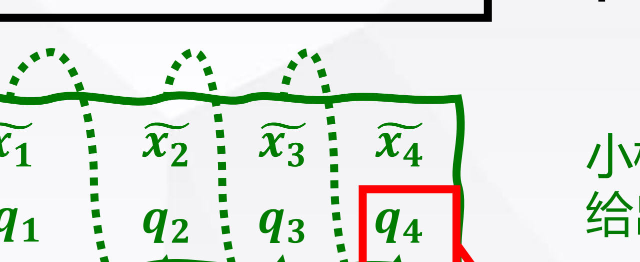
Speculative Decoding



[START] japan ' s benchmark nikkei 225 index rose 22 .6

x_1	x_2	x_3	… … …	x_t

prompt



小模型逐字生成，并给出每token的概率

p_1 p_2 p_3 p_4

大模型并行读入，一次forward生成每个token的概率

$$r < \min\left(1, \frac{p_i(\tilde{x}_i)}{q_i(\tilde{x}_i)}\right), r \sim U[0,1]$$

√ √ √ ×

通过比较两个模型给出的概率值来判断是否要采纳小模型的生成结果

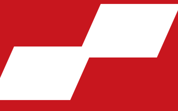
$$x_{t+c} \sim norm(\max(0, p_c - q_c))$$

x_4

在第一个被拒绝的token上，使用大模型按照右边的分布重新采样得到新的token

[1]. Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.
[2]. Chen, Charlie, et al. "Accelerating large language model decoding with speculative sampling." arXiv preprint arXiv:2302.01318 (2023).

Speculative Decoding



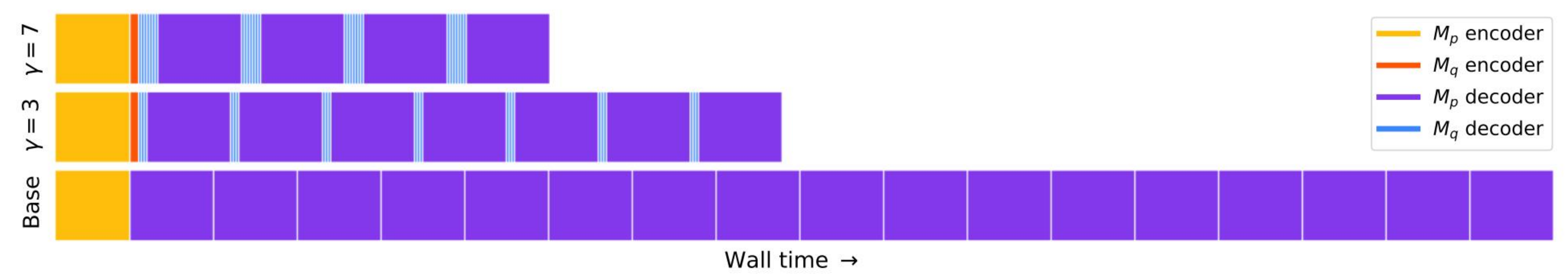
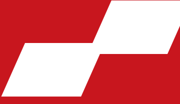


Figure 5. A simplified trace diagram for a full encoder-decoder Transformer stack. The top row shows speculative decoding with $\gamma = 7$ so each of the calls to M_p (the purple blocks) is preceded by 7 calls to M_q (the blue blocks). The yellow block on the left is the call to the encoder for M_p and the orange block is the call to the encoder for M_q. Likewise the middle row shows speculative decoding with $\gamma = 3$, and the bottom row shows standard decoding.

[1]. Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.
[2]. Chen, Charlie, et al. "Accelerating large language model decoding with speculative sampling." arXiv preprint arXiv:2302.01318 (2023).

Speculative Decoding



Speculative Decoding分别在翻译任务（ENDE）和文本摘要任务（CNNDM）上测试了加速效果

γ: 并行生成的Draft Token的数量

α: Draft Model和Target Model的相似度

$TEMP = 0$： Greedy Search
$TEMP = 1$： Speculative Sampling

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	**3.1X**
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	**2.3X**
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

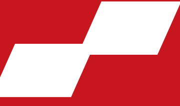
- Drafted Tokens的质量
- Verify的标准选择

- Draft Model M_p 与目标模型 M_q的对齐程度
- Draft Model M_p 自身的推理效率

SJTU

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." International Conference on Machine Learning. PMLR, 2023.

Speculative Decoding



Draft: Self-Drafting：避免部署多个模型

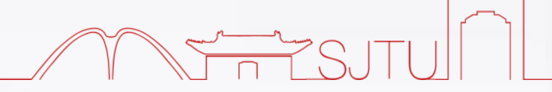
FFN Head
LM Head
FFN Head
Transformer Layer 4
Transformer Layer 3
Transformer Layer 2
Transformer Layer 1
Embedding

LM Head
Transformer Layer 4
Transformer Layer 3
Transformer Layer 2
Transformer Layer 1
Embedding

LM Head
Transformer Layer 4
MLP
Attention
Transformer Layer 3
MLP
Attention
Transformer Layer 2
MLP
Attention
Transformer Layer 1
MLP
Attention
Embedding

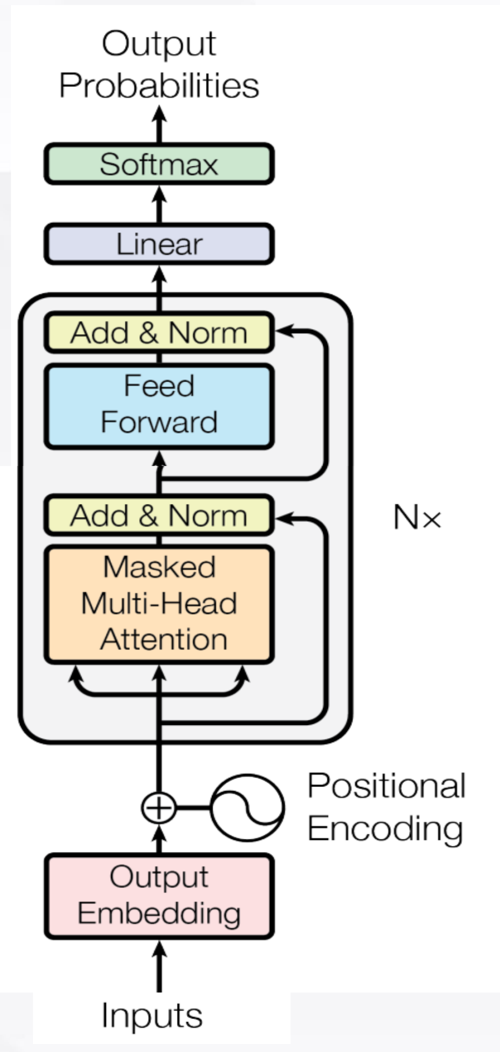
Blockwise Decoding、*Medusa* 在Transformer顶部增加FFN heads

利用 Early Exiting[1]、Layer Skipping[2] 进行draft tokens



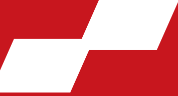
[1]. Yang, Nan, et al. "Inference with reference: Lossless acceleration of large language models." *arXiv preprint arXiv:2304.04487* (2023).
[2]. Zhang, Jun, et al. "Draft & verify: Lossless large language model acceleration via self-speculative decoding." *ACL*(2024).

总结



1. Attention机制层面
2. 逐层KV Cache压缩层面
3. 层间KV Cache压缩层面
4. 宏观计算架构层面

总结



1

- Attention机制层面

对attention机制本身所涉及的各个要素（如Q、K、V、softmax计算方式等）进行改进

2

- 逐层KV Cache压缩层面

对每一层的KV Cache序列长度进行压缩

3

- 层间KV Cache压缩层面

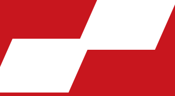
通过在不同层复用同一套KV Cache，来压缩显存开销

4

- 宏观计算架构层面

这一层面的方法并不改变模型本身，而是从模型所运行的计算环境的特点出发寻求优化方案

总结



1

• Attention机制层面

- Multi-Query Attention
- Group-Query Attention
- Multi-head Latent Attention
- Palu
- Performers

2

• 逐层KV Cache压缩层面

- Streaming LLM、LM-Infinite
- InfLLM
- H2O、Scissorhands、TOVA
- TreeKV
- SnapKV
- PyramidKV

3

• 层间KV Cache压缩层面

- Layer-Condensed KV
- Cross-Layer Attention
- You Only Cache Once
- Context Expansion with Parallel Encoding

4

• 宏观计算架构层面

- Flash Attention系列
- Flash Decoding系列
- Speculative Decoding

上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

谢谢大家！

饮 水 思 源　爱 国 荣 校