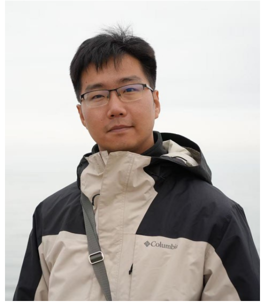
NEURAL INFORMATION PROCESSING SYSTEMS

FlashBias: Fast Computation of Attention with Bias

Haixu Wu[1], Minghao Guo[2], Yuezhou Ma[1], Yuanxu Sun[1], Jianmin Wang[1], Wojciech Matusik[2], Mingsheng Long[1]✉

[1]School of Software, Tsinghua University, [2]MIT CSAIL



Haixu Wu

Minghao Guo

Yuezhou Ma

Yuanxu Sun

Jianmin Wang

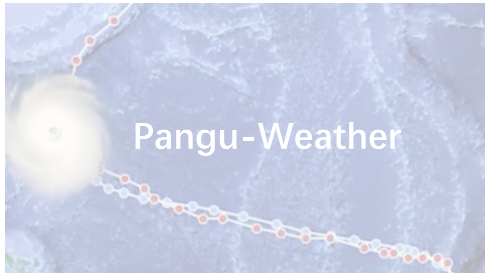
Wojciech Matusik

Mingsheng Long

Code Link: https://github.com/thuml/FlashBias

1.5x Speedup for Pairformer in AlphaFold 3; **2x** Speedup for Swin Transformer v2. **Try FlashBias!**

Attention in Advanced Language Models



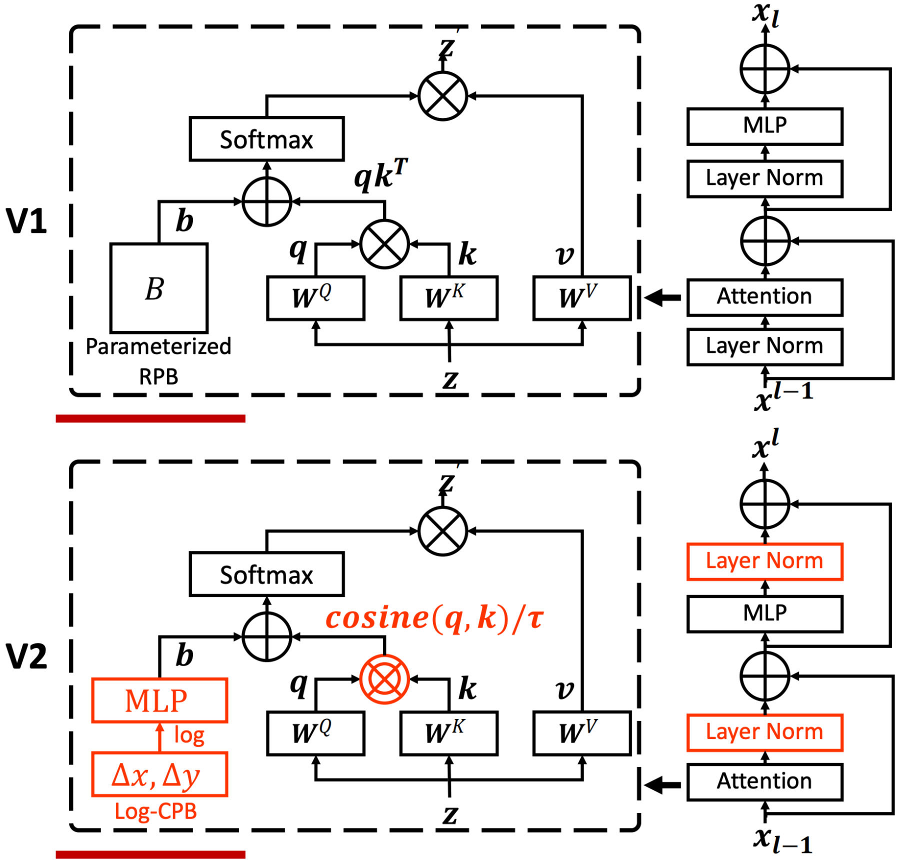
ALiBi Bias

$$\mathrm{softmax}(\mathbf{q}_i\mathbf{K}^\top + \underline{m\cdot[-(i-1), \ldots, -2, -1, 0]})$$

$$\begin{bmatrix} q_1\cdot k_1 & & & & \\ q_2\cdot k_1 & q_2\cdot k_2 & & & \\ q_3\cdot k_1 & q_3\cdot k_2 & q_3\cdot k_3 & & \\ q_4\cdot k_1 & q_4\cdot k_2 & q_4\cdot k_3 & q_4\cdot k_4 & \\ q_5\cdot k_1 & q_5\cdot k_2 & q_5\cdot k_3 & q_5\cdot k_4 & q_5\cdot k_5 \end{bmatrix} + \begin{bmatrix} 0 & & & & \\ -1 & 0 & & & \\ -2 & -1 & 0 & & \\ -3 & -2 & -1 & 0 & \\ -4 & -3 & -2 & -1 & 0 \end{bmatrix} \bullet m$$

Press et al., Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation, ICLR 2022

Attention in Advanced Vision Models



$$\mathrm{SoftMax}(QK^T/\sqrt{d} + B)V,$$

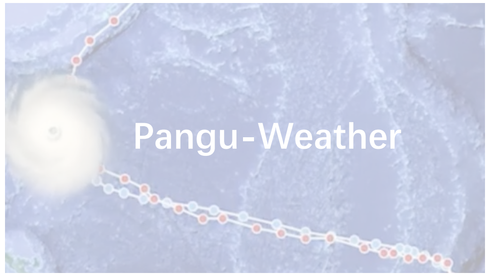
Relative Position Bias

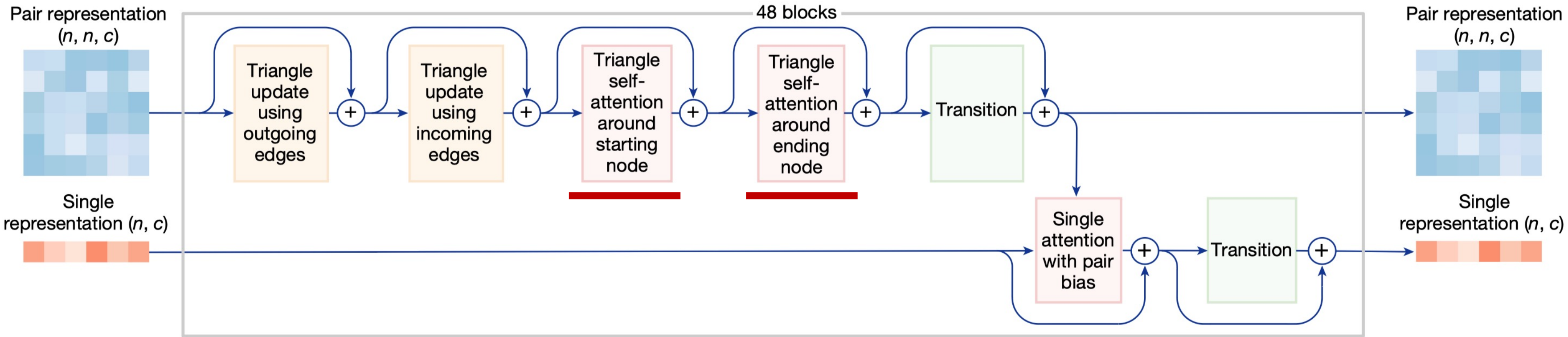
$$\cos(\mathbf{q}_i, \mathbf{k}_j)/\tau + B_{ij}$$

Relative Position Bias

Liu et al., Swin Transformer V2: Scaling Up Capacity and Resolution, CVPR 2022

Attention in Advanced Scientific Models





 # Attention

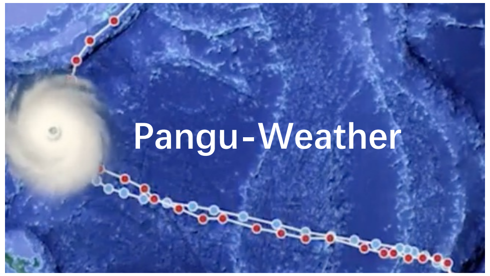
$$5:\ a_{ijk}^{h} = \mathrm{softmax}_k\left(\frac{1}{\sqrt{c}}\,\mathbf{q}_{ij}^{h\top}\mathbf{k}_{ik}^{h} + \underline{b_{jk}^{h}}\right)$$

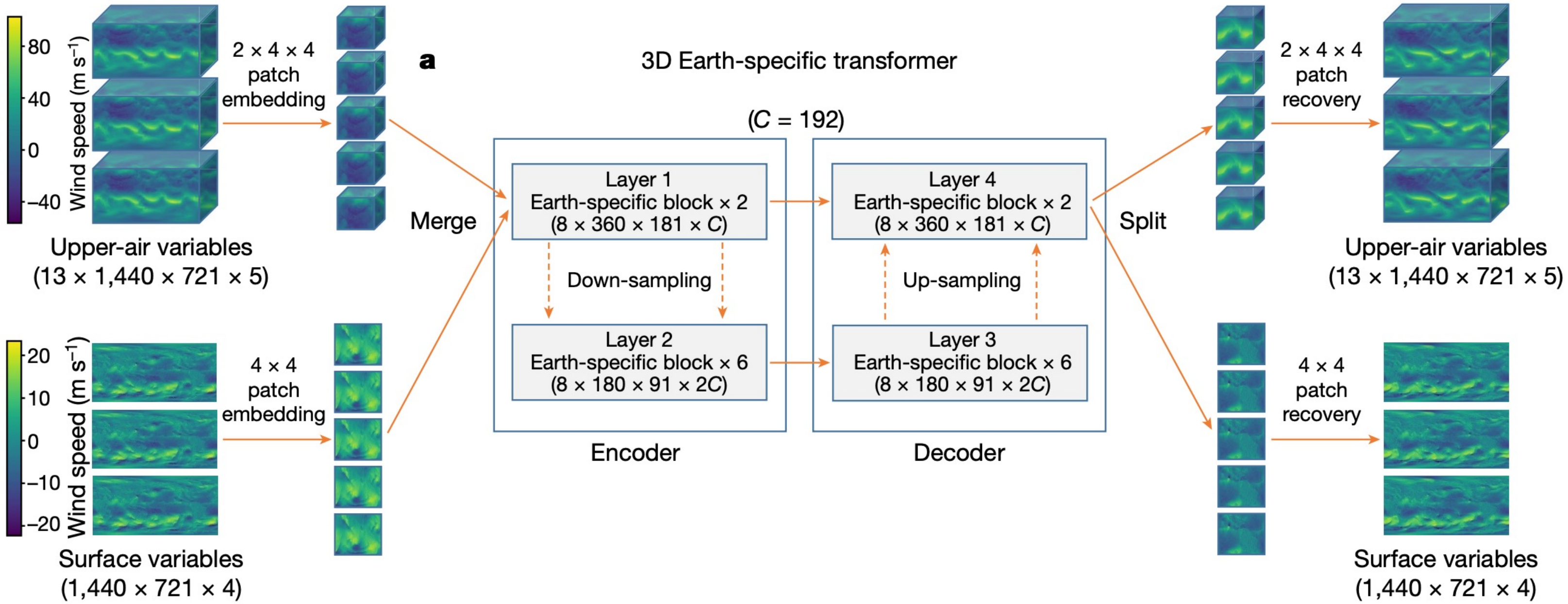
$$6:\ \mathbf{o}_{ij}^{h} = \mathbf{g}_{ij}^{h} \odot \sum_k a_{ijk}^{h}\mathbf{v}_{ik}^{h}$$

Pair Representation Bias

Abramson et al., Accurate structure prediction of biomolecular interactions with AlphaFold 3, Nature 2024

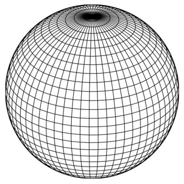
Attention in Advanced Scientific Models





$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{SoftMax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{D} + \underline{\mathbf{B}})\mathbf{V}$$

Earth-specific Positional Bias



Bi et al., Accurate medium-range global weather forecasting with 3D neural networks, Nature 2023

Attention with Bias

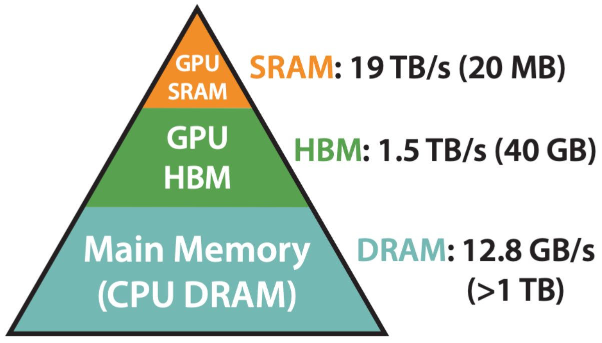
$$\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} + \mathbf{b})\mathbf{v}.$$

queries $\mathbf{q} \in \mathbb{R}^{N\times C}$, keys $\mathbf{k} \in \mathbb{R}^{M\times C}$ and values $\mathbf{v} \in \mathbb{R}^{M\times C}$, bias $\mathbf{b} \in \mathbb{R}^{N\times M}$

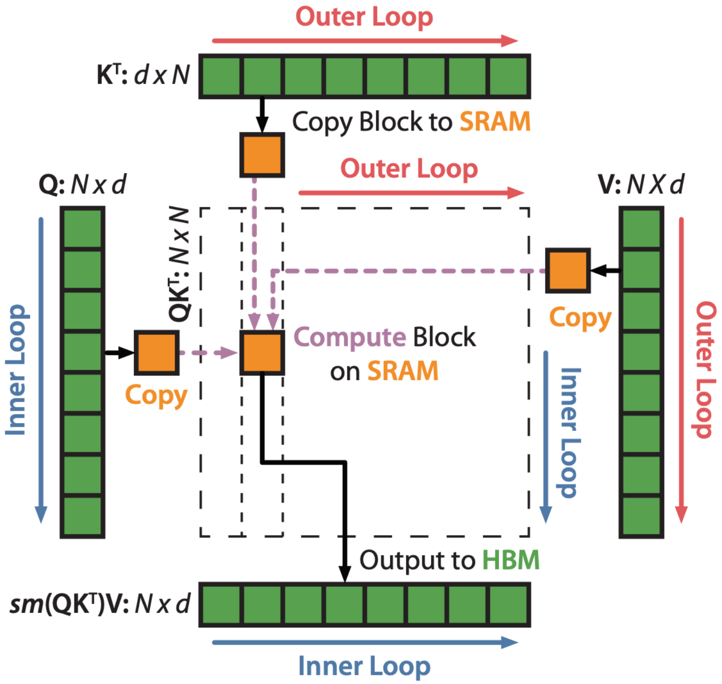
Introduce prior knowledge to guide attention learning

Vanilla FlashAttention

$$\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} + \underline{\mathbf{b}})\mathbf{v}.$$

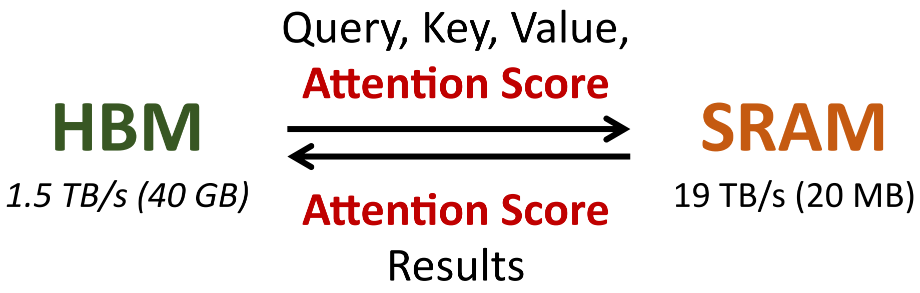


Memory Hierarchy with Bandwidth & Memory Size

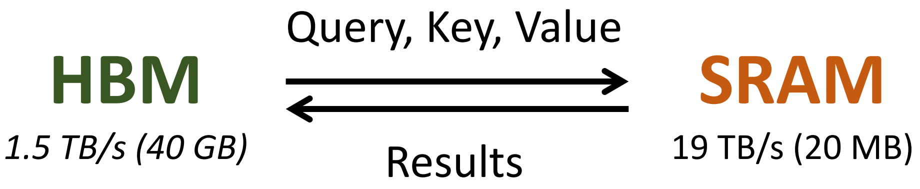


FlashAttention

- ➢ Standard Implementation: Quadratic IO Complexity



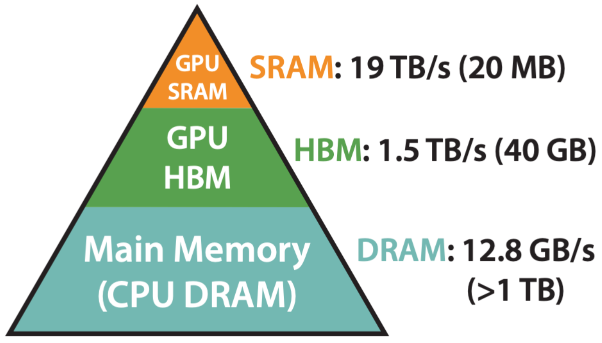
- ➢ FlashAttention: Reduced IO Complexity



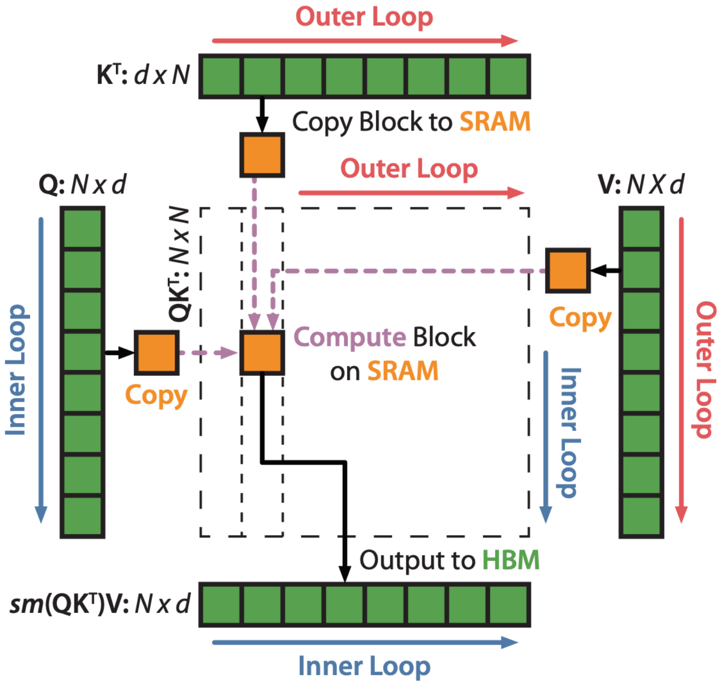
Dao et al., FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, NeurIPS 2022
Dao et al., FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning, ICLR 2024

Vanilla FlashAttention Fails in Attention with Bias

$$\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} + \underline{\mathbf{b}})\mathbf{v}.$$

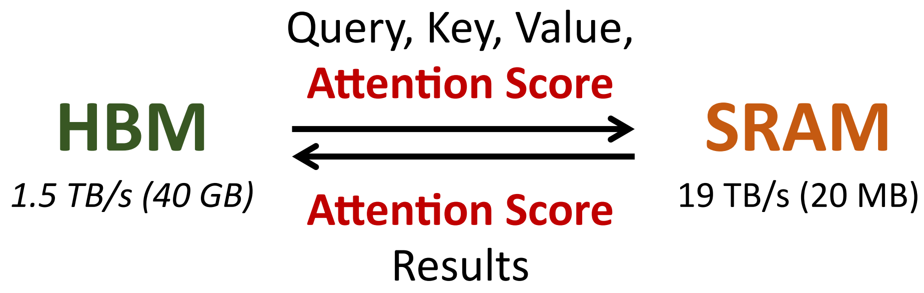


Memory Hierarchy with Bandwidth & Memory Size

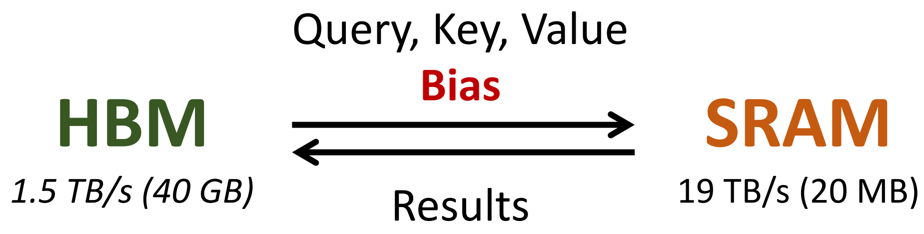


FlashAttention

- Standard Implementation: **Quadratic IO Complexity**



- FlashAttention: **Quadratic IO Complexity**



Dao et al., FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, NeurIPS 2022

Dao et al., FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning, ICLR 2024

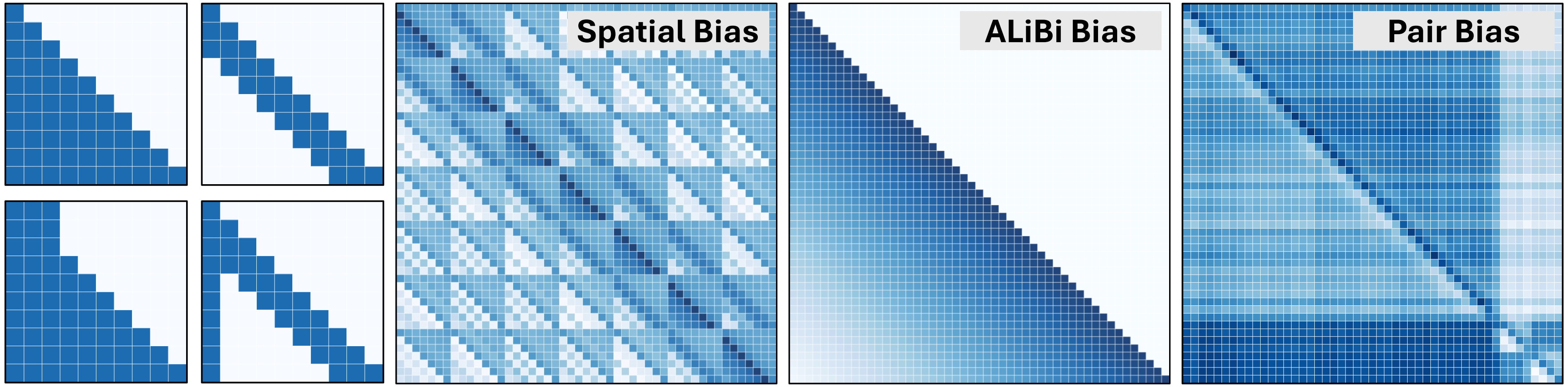
Challenge in Optimizing Attention with Bias

$$\mathbf{o} = \mathrm{softmax}\left(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} + \underline{\mathbf{b}}\right)\mathbf{v}.$$

queries $\mathbf{q} \in \mathbb{R}^{N\times C}$, keys $\mathbf{k} \in \mathbb{R}^{M\times C}$ and values $\mathbf{v} \in \mathbb{R}^{M\times C}$, bias $\underline{\mathbf{b} \in \mathbb{R}^{N\times M}}$

Introduce prior knowledge to guide attention learning

Sparse

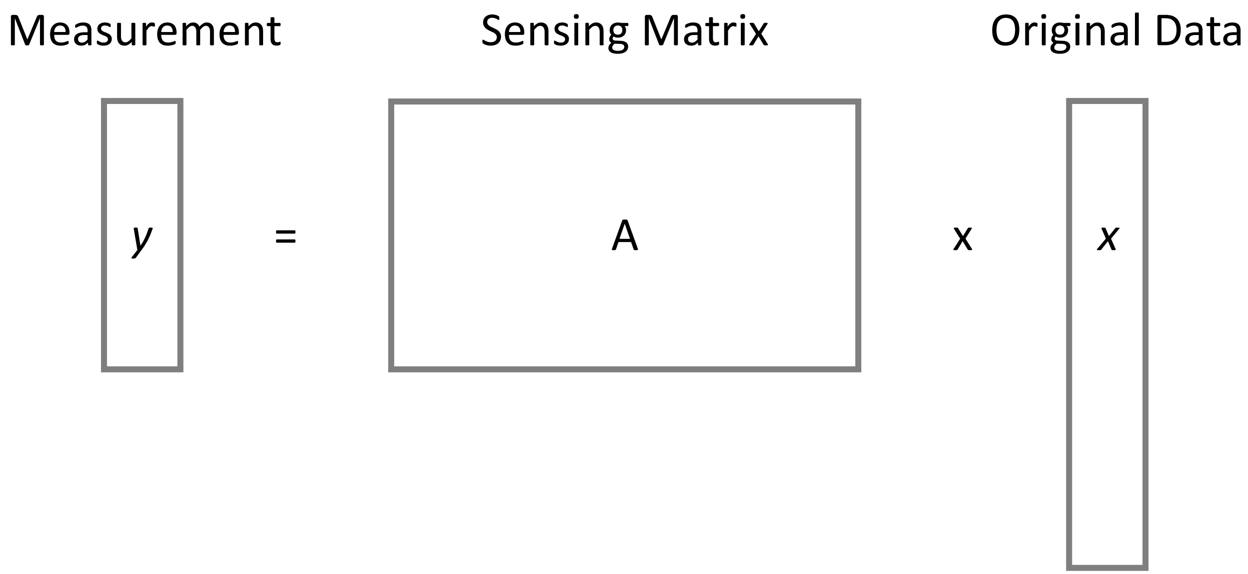


(a) Attention Mask (b) Attention Bias in Famous Transformers

Dense
Continuous

Inevitable IO complexity for loading the dense bias matrix

A Typical Compressed Sensing Problem



- **Compressed Sensing:** “measurement” (storage) is expensive, but the computation is cheap
- **Attention Computation:** IO is slow, but on-chip computation is fast

<u>If we can compress the original data (Bias Matrix), we can reduce the IO complexity.</u>

Donoho et al., Compressed sensing. IEEE Transactions on information theory 2006

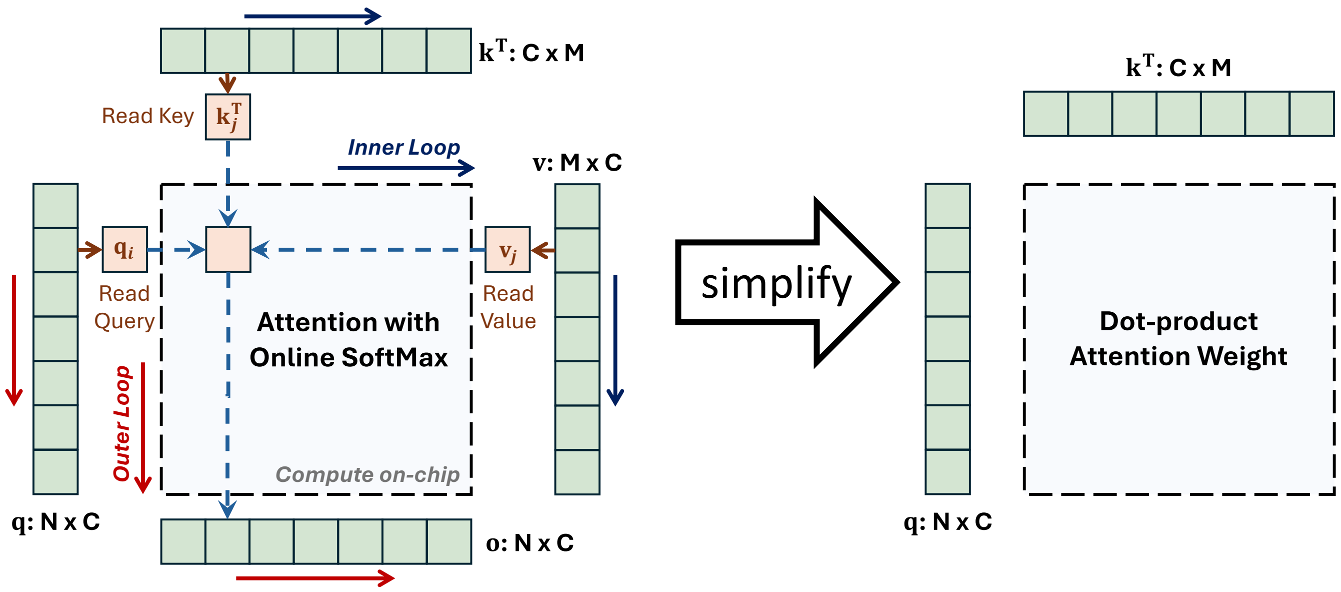
Why FlashAttention is Fast? Underlying low rank assumption

Given Sequence len N, Channel dim C, SRAM size S and $C=\alpha N$, $S=\beta NC$

1) FlashAttention IO Complexity is $\Theta\left(\left(1+\frac{1}{\alpha}\right)\beta\right)$ *smaller than standard attention*

2) Suppose dot-product attention weight $\mathbf{s} = \mathbf{q}\mathbf{k}^{\mathrm{T}}$ *is of rank R,* $\alpha \geq \frac{R}{N}$

The speedup ratio of FlashAttention $\propto \frac{1}{\alpha}$ and $\propto \beta$. β is usually fixed. **α determines performance.**



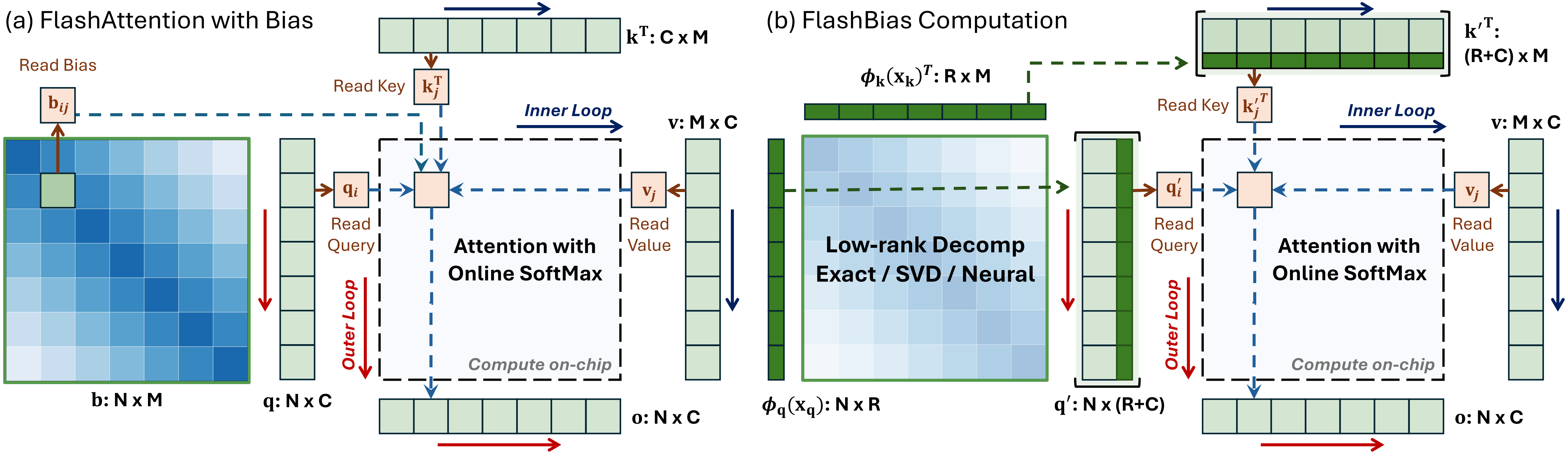
Reverse thinking 💡

Query and key are from low-rank decomposition of attention score.

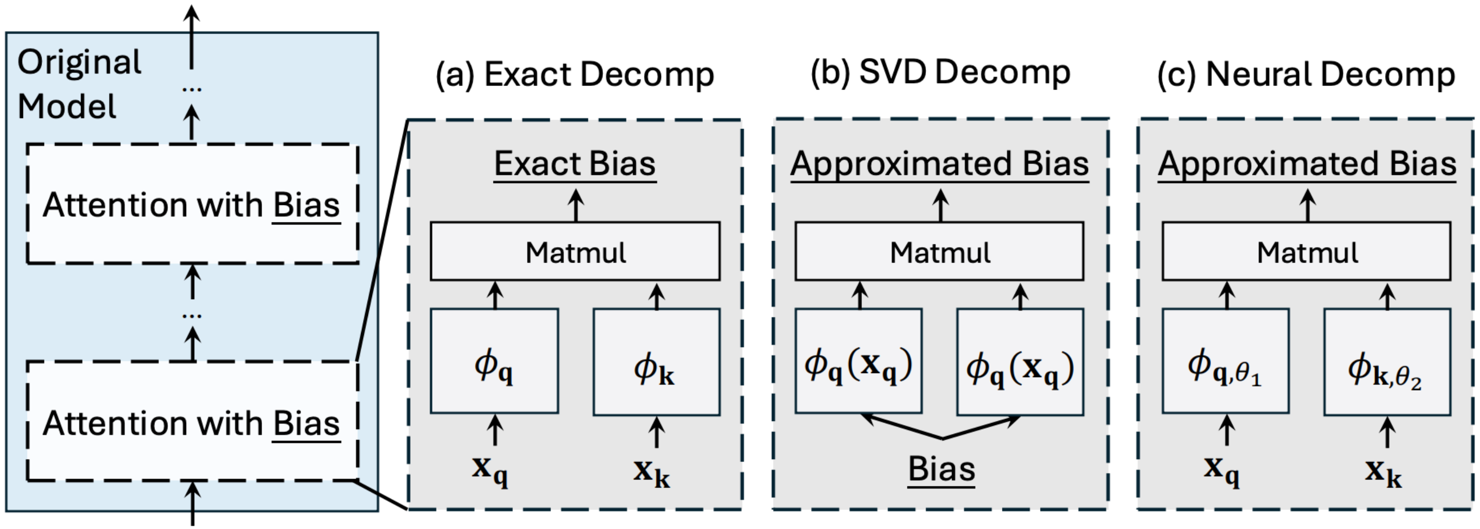
FlashBias: Achieving theoretically optimal complexity

1) Low-rank Decomp $\mathbf{b} = f(\mathbf{x_q}, \mathbf{x_k}) = \phi_{\mathbf{q}}(\mathbf{x_q})\phi_{\mathbf{k}}(\mathbf{x_k})^\top, \ \ \phi_{\mathbf{q}}, \phi_{\mathbf{k}} : \mathbb{R}^{C'} \to \mathbb{R}^{R}.$

2) Fast computation $\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} + \mathbf{b})\mathbf{v} = \mathrm{softmax}\left(\frac{[\mathbf{q}|\sqrt{C}\phi_{\mathbf{q}}(\mathbf{x_q})]\ [\mathbf{k}|\phi_{\mathbf{k}}(\mathbf{x_k})]^\top}{\sqrt{C}}\right)\mathbf{v}$



FlashBias: Three concrete instantiations for decomposition



Domain	Bias / Model	Type
Language	ALiBi [29]	(a)
Vision	Swin Trans. [24]	(b)
Science	Spatial Distance	(a)
	Pangu-Weather [4]	(b)
	AlphaFold [1]	(c)

1) Exact Decomp: for some representative bias, such as ALiBi or spatial distance bias.

$$f(\mathbf{x}_{\mathbf{q},i}, \mathbf{x}_{\mathbf{k},j}) = i - j \;,\; \phi_{\mathbf{q}}(\mathbf{x}_{\mathbf{q},i}) = [1, i] \textit{ and } \phi_{\mathbf{k}}(\mathbf{x}_{\mathbf{k},j}) = [-j, 1]$$

2) SVD Decomp: when the bias term is learnable model parameters

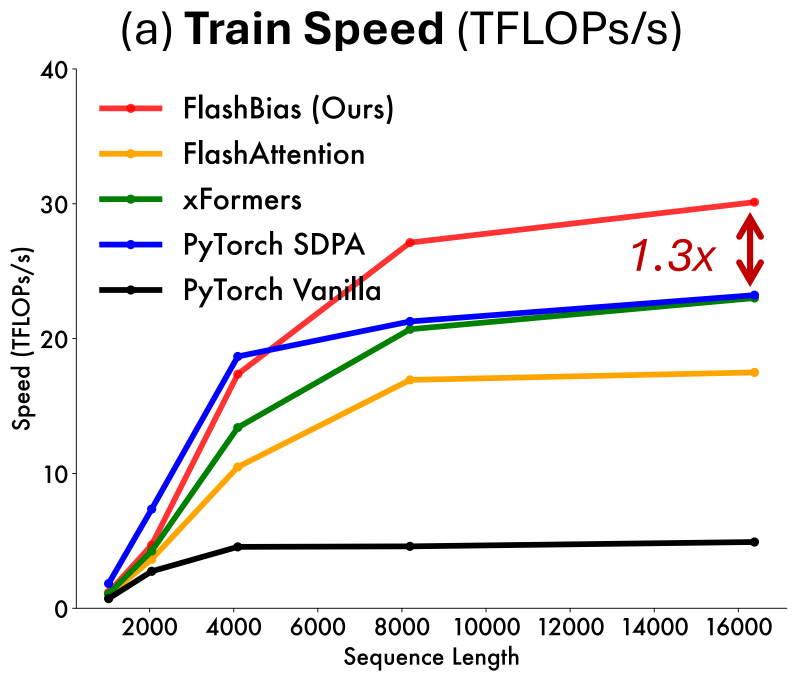
3) Neural Decomp: when the bias term is data dependent

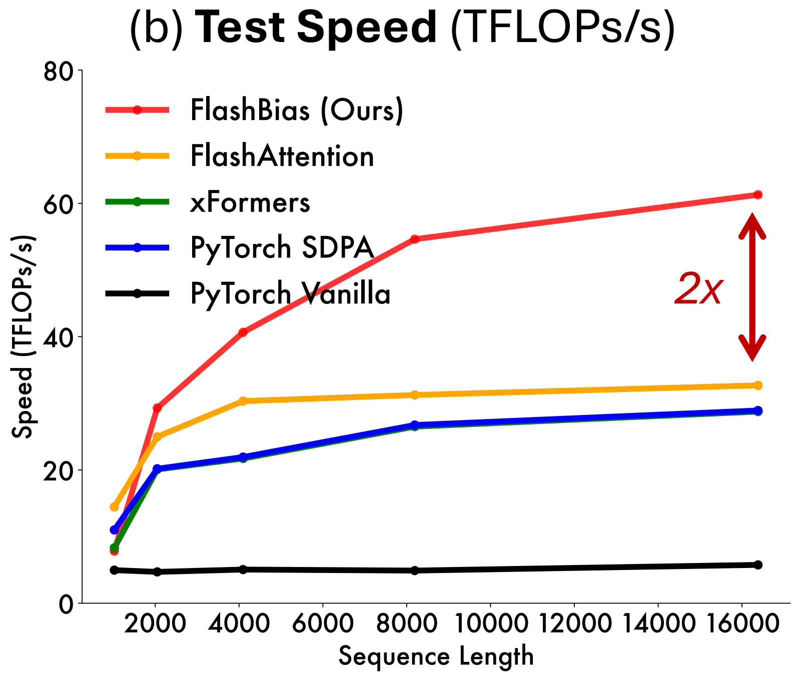
$$\min_{\theta_1,\theta_2} \mathcal{L}(\mathbf{x_q}, \mathbf{x_k}) = \|\widehat{\phi}_{\mathbf{q},\theta_1}(\mathbf{x_q})\widehat{\phi}_{\mathbf{k},\theta_2}(\mathbf{x_k})^\top - f(\mathbf{x_q}, \mathbf{x_k})\|_2^2.$$

Relative information

Usage and Comparison

```
>> from flash_bias_triton import flash_bias_func
>> output = flash_bias_func(q, k, v, q_bias, k_bias, mask=None, causal=False, softmax_scale=1/math.sqrt(headdim))
```





Try FlashBias!

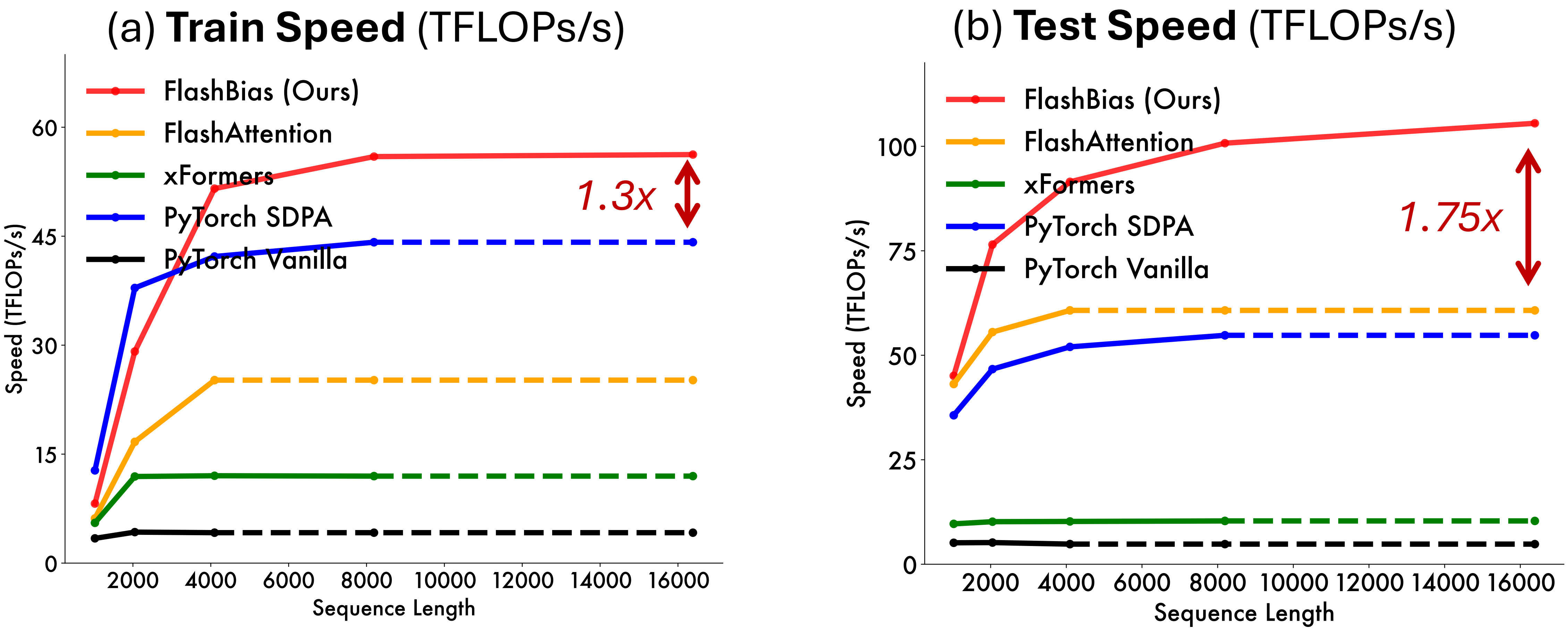
Surpass FlashAttention, PyTorch SDPA, xFormers (bs2-head4-headdim32-noncausal-rank8)

https://github.com/Dao-AILab/flash-attention

https://github.com/facebookresearch/xformers

https://docs.pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html

Case 1: GPT-2 with ALiBi Bias (Exact Decomp, Causal Mask, R=2)



batchsize1-head50-headdim32-causal-rank2

Case 2: Swin Transformer V2 (SVD Decomp, R=16)

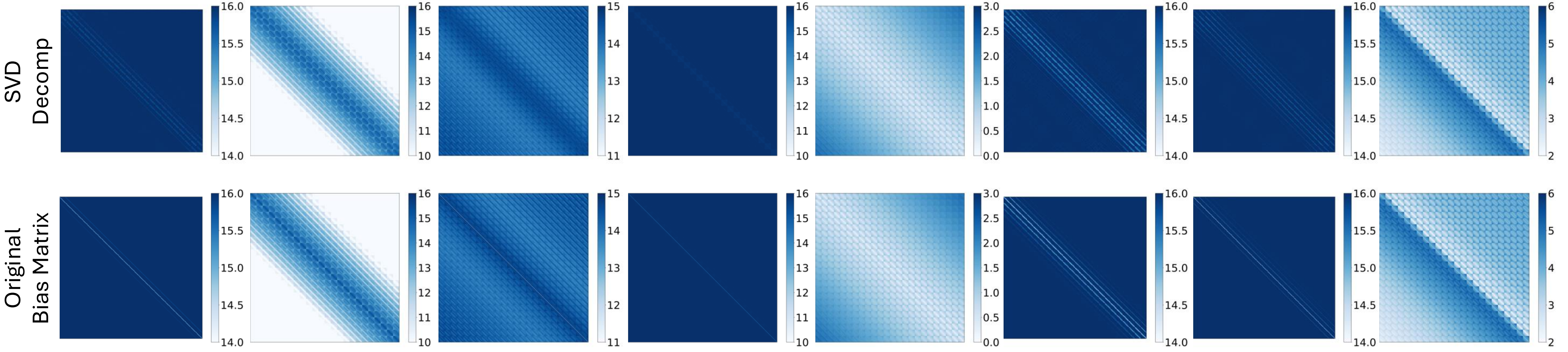
Table 4: Experiment of SwinV2-B on ImageNet-1K. #Time and #Mem correspond to inference efficiency on A100 per batch. Offline calculation of SVD for all biases takes 4.79s.

Method	Acc@1	Acc@5	Time(s)	Mem(MB)
Official Code	87.144%	98.232%	0.473	12829
Pure FlashAttention	9.376%	19.234%	0.180	3957
FlashAttention with Bias	87.142%	98.232%	0.230	11448
FlexAttention [11]	87.142%	98.232%	2.885	25986
INT8 PTQ	86.46%	*Around 22% speed up*		
FlashBias (Ours)	87.186%	98.220%	0.190	9429

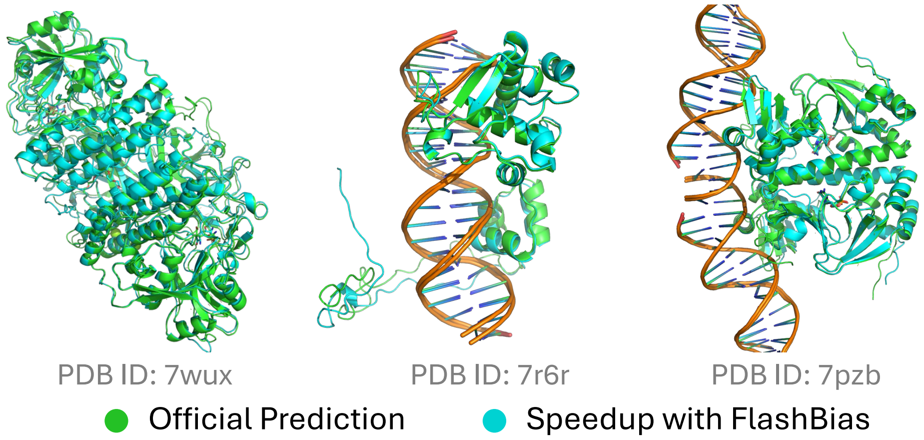
Inference time: 0.473s → 0.190s (60% reduction)

GPU memory: 12829MB → 9429MB (27% reduction)

2x speedup without any loss of accuracy



Case 3: AlphaFold 3 (Neural Decomp, R=96)



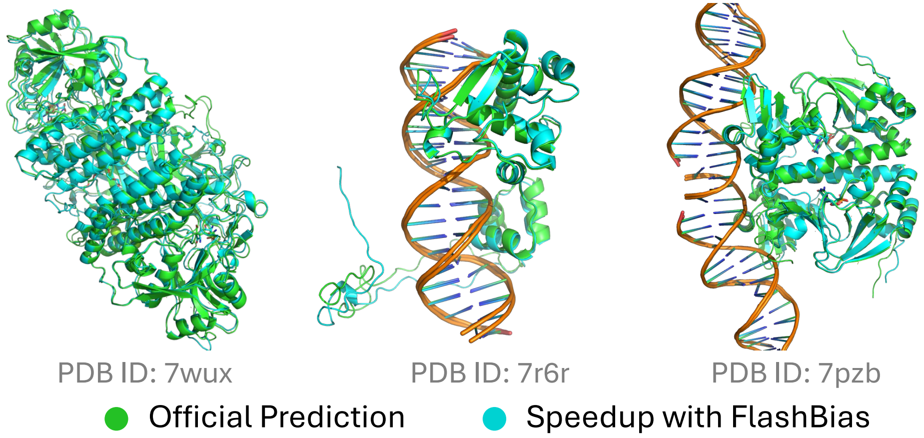
Method	Test Set	PDB ID 7wux		
	pLLDDT Loss ↓	pTM ↑	Time(s)	Mem(GB)
Open-sourced Code	3.3724	0.9500	26.85	13.62
FlashAttention w/o Bias	4.3669	0.1713	8.27	12.89
FlashAttention w/ Bias	3.3724	0.9500	20.39	13.62
FlashBias (Ours)	3.3758	0.9498	18.19	13.62

Inference time: 26.85s → 18.19s (32% reduction)

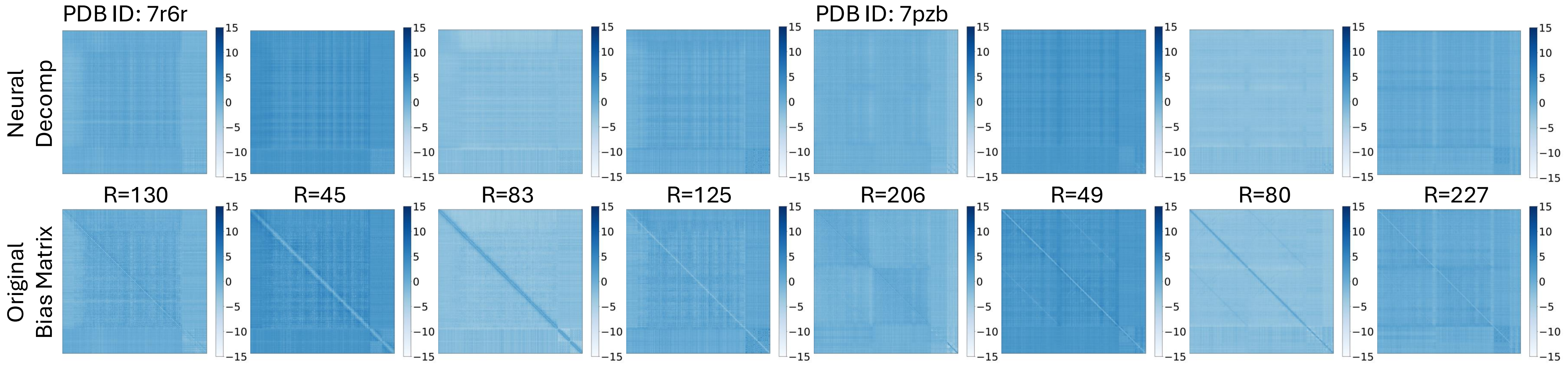
1.5x speedup without any loss of accuracy

https://github.com/bytedance/Protenix

Case 3: AlphaFold 3 (Neural Decomp, R=96)

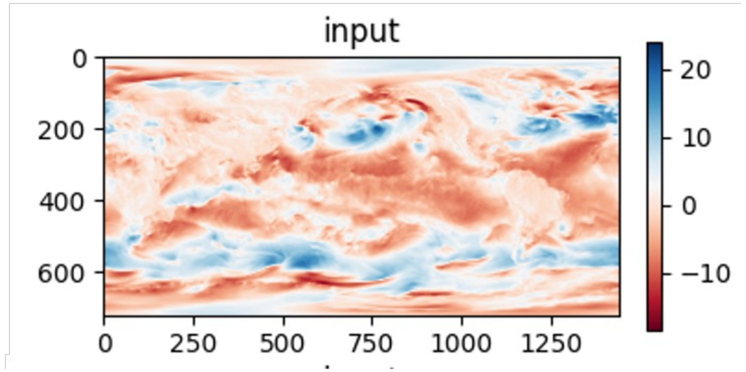


Method	Test Set	PDB ID 7wux		
	pLLDDT Loss ↓	pTM ↑	Time(s)	Mem(GB)
Open-sourced Code	3.3724	0.9500	26.85	13.62
FlashAttention w/o Bias	4.3669	0.1713	8.27	12.89
FlashAttention w/ Bias	3.3724	0.9500	20.39	13.62
FlashBias (Ours)	3.3758	0.9498	18.19	13.62



https://github.com/bytedance/Protenix

Case 4: Pangu-Weather (SVD Decomp, R=56)



Method	Output Difference	Time(s/100iters)	Mem(MB)
Open-sourced Code	-	98.022	26552
FlashAttention w/o bias	0.0128	74.089	12141
FlashAttention w/ bias	-	79.649	13186
FlashBias (Ours)	0.0003	**76.779**	**12222**

Inference time: 98s → 77s (21% reduction)

GPU memory: 26552MB → 12222MB (54% reduction)

speedup without any loss of accuracy

https://github.com/zhaoshan2/pangu-pytorch

Extension to Multiplicative Bias

$$\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} \odot \underline{\mathbf{b}})\mathbf{v}$$

queries $\mathbf{q} \in \mathbb{R}^{N\times C}$, keys $\mathbf{k} \in \mathbb{R}^{M\times C}$ and values $\mathbf{v} \in \mathbb{R}^{M\times C}$, bias $\underline{\mathbf{b} \in \mathbb{R}^{N\times M}}$

Introduce prior knowledge to guide attention learning

FlashBias' extension to multiplicative bias:

$$\mathbf{o} = \mathrm{softmax}(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{C}} \odot \mathbf{b})\mathbf{v} = \mathrm{softmax}(\frac{\mathbf{q}'\mathbf{k}'^\top}{\sqrt{C}})\mathbf{v},$$

where $\mathbf{q}' = [\mathbf{q} \odot \phi_{\mathbf{q},1}, \cdots, \mathbf{q} \odot \phi_{\mathbf{q},R}] \in \mathbb{R}^{N\times CR}$, $\mathbf{k}' = [\mathbf{k} \odot \phi_{\mathbf{k},1}, \cdots, \mathbf{k} \odot \phi_{\mathbf{k},R}] \in \mathbb{R}^{N\times CR}$.

Example: $\mathbf{b}_{ij} = \cos(i-j)$

$$\phi_{\mathbf{q}}(\mathbf{x}_{\mathbf{q},i}) = [\cos(i), \sin(i)] \in \mathbb{R}^{1\times 2}, \ \ \phi_{\mathbf{k}}(\mathbf{x}_{\mathbf{k},j}) = [\cos(j), \sin(j)] \in \mathbb{R}^{1\times 2}.$$

NEURAL INFORMATION PROCESSING SYSTEMS

Thank You!

wuhaixu98@gmail.com

Code Link: https://github.com/thuml/FlashBias

1.5x Speedup for Pairformer in AlphaFold 3; **2x** Speedup for Swin Transformer v2.

Try FlashBias!