FlashAttention图解（如何加速Attention）

最新FlashDecoding++

Austin：【FlashAttention-V4，非官方】FlashDecoding++

FlashAttention V2和V3版本详解：

Austin：FlashAttention2详解（性能比FlashAttention提升200%）

Austin：FlashAttenion-V3: Flash Decoding详解

Motivation

当输入序列（sequence length）较长时，Transformer的计算过程缓慢且耗费内存，这是因为self-attention的time和memory complexity会随着sequence length的增加成二次增长。

标准Attention的中间结果$\mathbf{S}, \mathbf{P}$（见下文）通常需要通过高带宽内存（HBM）进行存取，两者所需内存空间复杂度为$O(N^2)$。本文分析：

- FlashAttention: 对HBM访问的次数为$O(N^2d^2M^{-1})$
- Attention: 对HBM访问的次数为$\Omega\left(Nd+N^2\right)$

往往$N \gg d$（例如GPT2中N=1024，d=64），因此FlashAttention会快很多。下图展示了两者在GPT-2上的Forward+Backward的GFLOPs、HBM、Runtime对比（A100 GPU）：

Attention	Standard	FLASHATTENTION
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

GPU中存储单元主要有HBM和SRAM：HBM容量大但是访问速度慢，SRAM容量小却有着较高的访问速度。例如：A100 GPU有40-80GB的HBM，带宽为1.5-2.0TB/s；每108个流式多核处理器各有192KB的片上SRAM，带宽估计约为19TB/s。可以看出，片上的SRAM比HBM快一个数量级，但尺寸要小许多数量

级。

综上，FlashAttention目的不是节约FLOPs，而是减少对HBM的访问。重点是FlashAttention在训练和预测过程中的结果和标准Attention一样，对用户是无感的，而其他加速方法做不到这点。

阅读本文需要了解的符号定义：

- N: sequence length
- d: head dimension
- M: the size of SRAM
- Ω：大于等于的数量级复杂度
- O：小于等于的数量级复杂度
- Θ：同数量级的复杂度
- o：小于的数量级复杂度

Method

Attention

标准Attention输入为$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$，输出为$\mathbf{O} \in \mathbb{R}^{N \times d}$，计算如下：

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \mathrm{softmax}(\mathbf{S}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

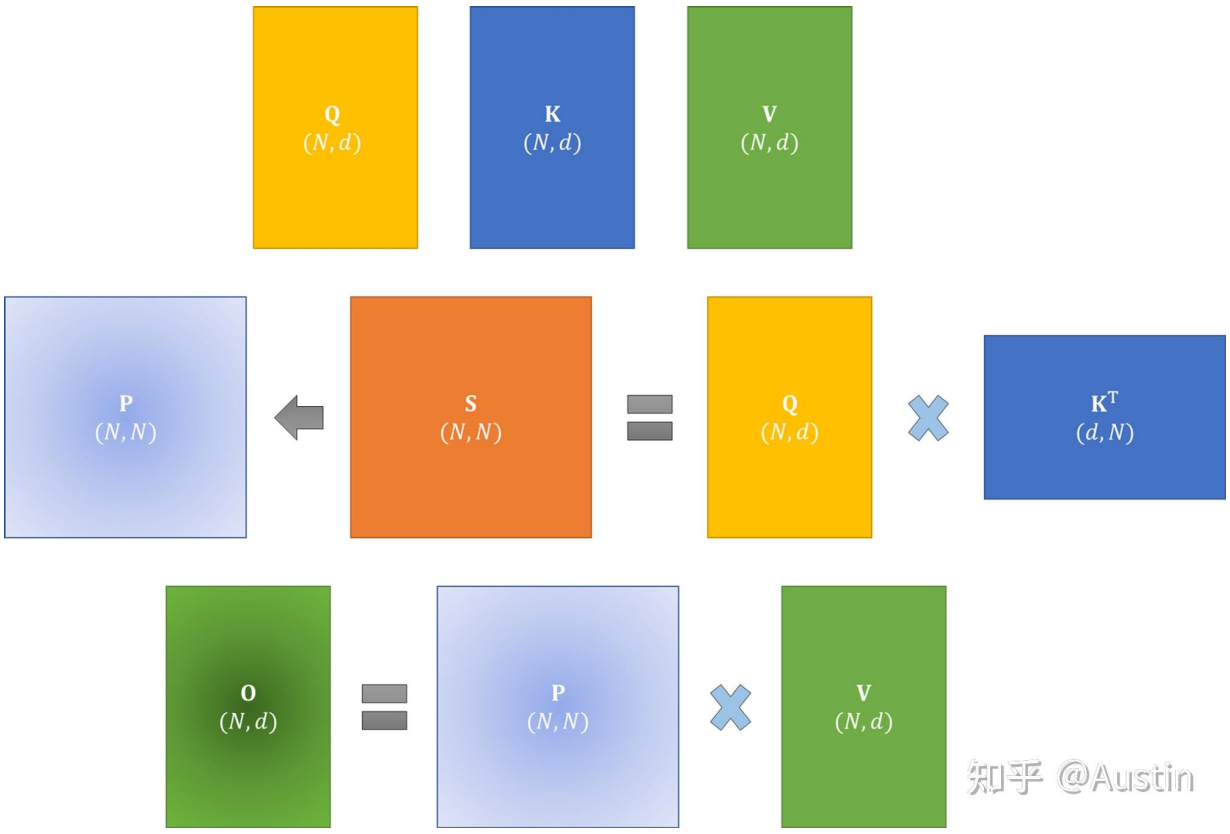
其中softmax操作是row-wise的，即每行都算一次softmax，一共计算N行。

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

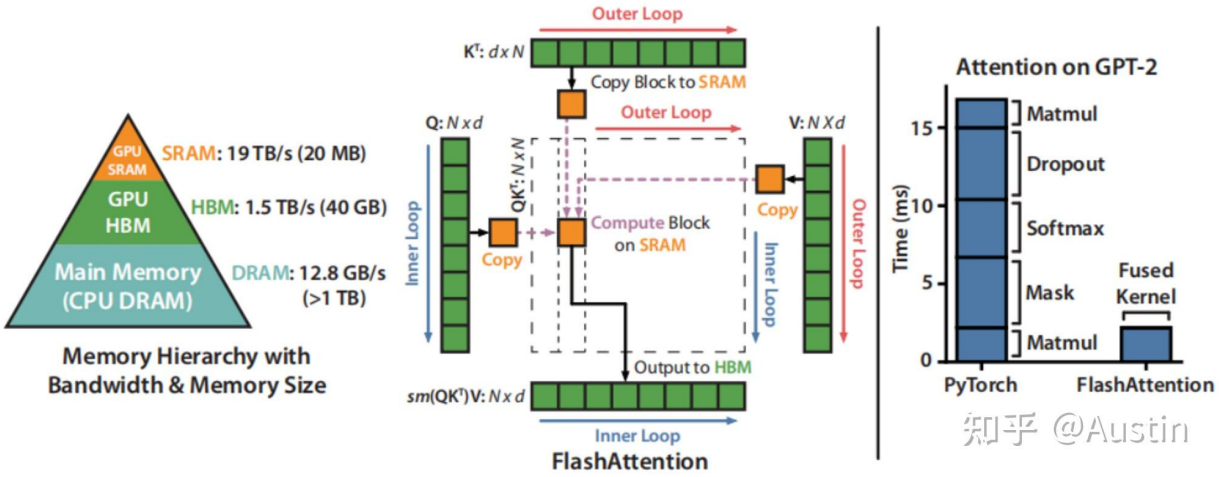
1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

计算流程图如下：



FlashAttention

建议先阅读这篇知乎文章，重复内容不再赘述。



Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.

for $1 \le j \le T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \le i \le T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.

    On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.

    Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.

    Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.

  end for

end for

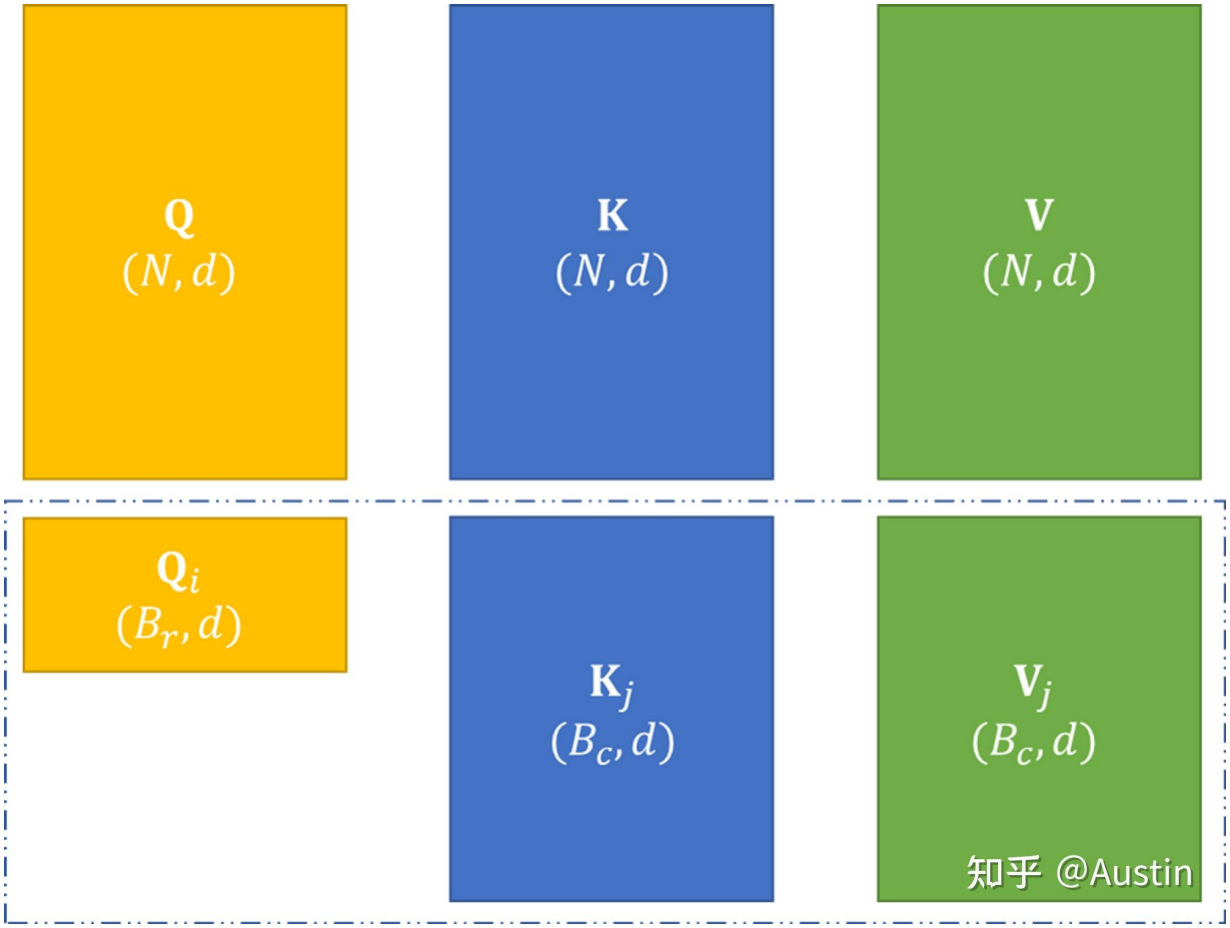
Return $\mathbf{O}$.

以GPT2和A100为例：A100的SRAM大小为192KB=196608B；GPT2中$N = 1024, d = 64$，对应的$\mathbf{Q}, \mathbf{K}, \mathbf{V}$的维度为$N \times d = 1024 \times 64$，中间结果$\mathbf{S}, \mathbf{P}$的维度为$N \times N = 1024 \times 1024$。本例中FlashAttention的参数为：

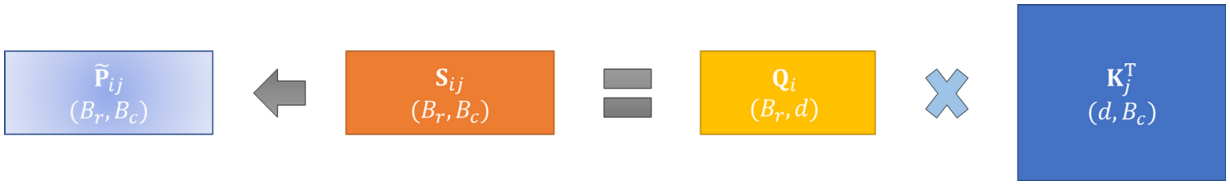
- $B_c = \lceil 196608/4/64 \rceil = 768$；$B_r = \min(768, 64) = 64$
- $T_c = \lceil 1024/768 \rceil = 2$；$T_r = \lceil 1024/64 \rceil = 16$

对应的计算过程：

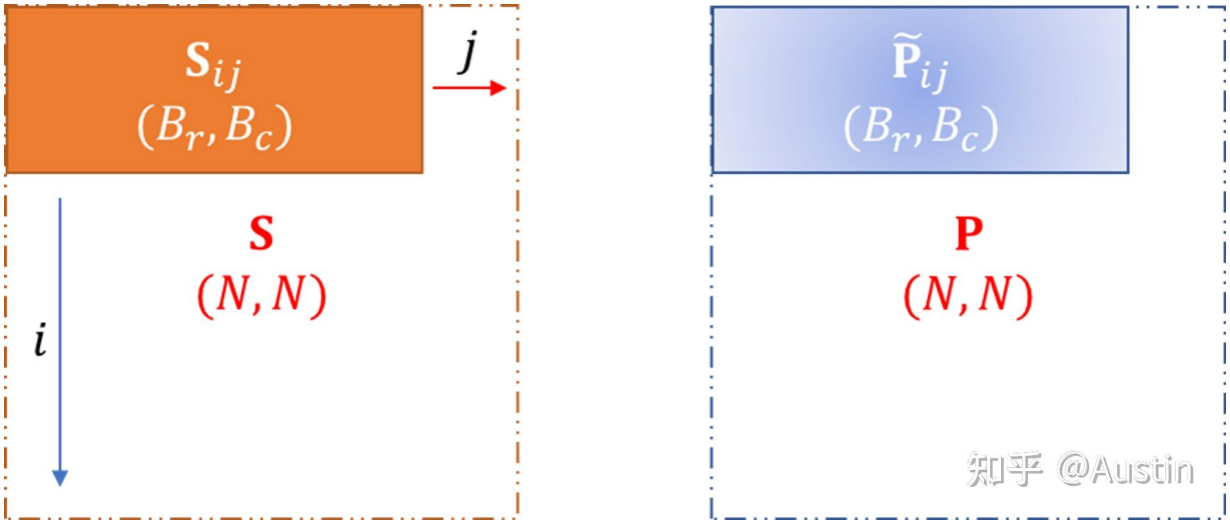
- 每次外循环（outer loop，j）载入的$\mathbf{K}_j, \mathbf{V}_j$的大小为$B_c \times d = 768 \times d$，一共循环$T_c = 2$次



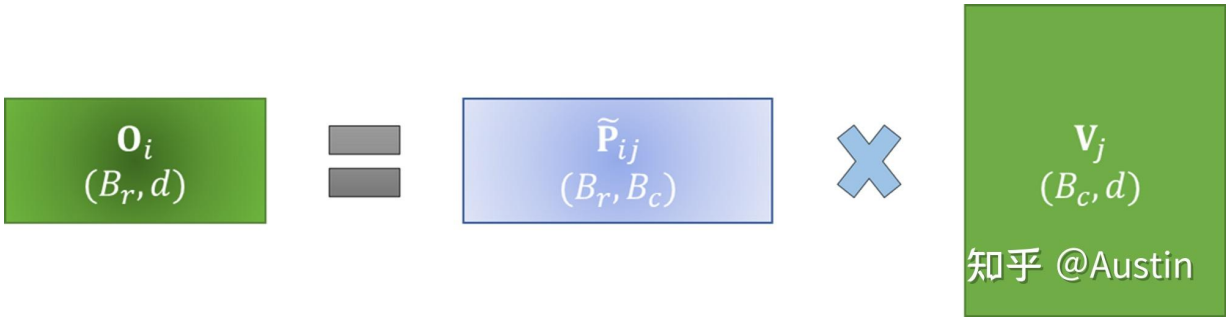
- 每次内循环（inner loop，i）载入的$\mathbf{Q}_i, \mathbf{O}_i$的大小为$B_r \times d = 64 \times d$，一共循环$T_r = 16$次（总次数还需要乘以外循环）
- $\mathbf{S}_{ij} = \mathbf{Q}_i \times \mathbf{K}_j^T$，即为（下标表示维度）：$C_{64\times768} = A_{64\times d} \times B_{d\times768}$。
- Error: '_' allowed only in math mode，$\tilde{\mathbf{P}}_{ij}$表示和标准attention计算的$\mathbf{P}_{ij}$有区别，因为Error: '_' allowed only in math mode得到的最大值可能不是$\mathbf{S}$第i行的最大值。$\tilde{\mathbf{P}}_{ij}$的大小和$\mathbf{S}_{ij}$一样，都为$B_r \times B_c = 64 \times 768$。



- $\tilde{\mathbf{P}}_{ij}$和$\mathbf{S}_{ij}$只是部分结果，如下图所示，外循环j是横向（特征维d）移动的，内循环i是纵向（序列维N）移动的。**换句话说，外循环在顺序计算特征，内循环在顺序计算序列。**



- $\mathbf{O}_i$的大小为$B_r \times d$，第二维d是满的（和最终$\mathbf{O}$一样），**这意味着每次外循环都要重新更新当前批次中的特征**，即虽然第一次外循环$\tilde{P}_{00} \times V_0$和第二次外循环$\tilde{P}_{01} \times V_1$都会得到$\mathbf{O}_0$，但是第二次$\mathbf{O}_0$的是基于第一次$\mathbf{O}_0$重新生成的。



- $\mathrm{diag}(\dots)$作用是将vector生成为一个对角矩阵，从而实现相同长度的两个vector进行element-wise相乘。

Theorem 1. FlashAttention的FLOPs为 $O(N^2 d)$，除了input和output，额外需要的内存为 $O(N)$。

- Theorem 1的证明过程如下。
  影响FLOPs的主要是matrix multiplication。在一次循环中：
- Algorithm 1第9行：计算$\mathbf{Q}_i \mathbf{K}_j^\top \in \mathbb{R}^{B_r \times B_c}$。由于$\mathbf{Q}_i \in \mathbb{R}^{B_r \times d}$，$\mathbf{K}_j \in \mathbb{R}^{B_c \times d}$，因此一次计算需要的FLOPs为$O(B_r B_c d)$。
- Algorithm 1第12行：计算$\tilde{\mathbf{P}}_{ij} \mathbf{V}_j \in \mathbb{R}^{B_r \times d}$。由于$\tilde{\mathbf{P}}_{ij} \in \mathbb{R}^{B_r \times B_c}$，$\mathbf{V}_j \in \mathbb{R}^{B_c \times d}$，因此一次计算需要的FLOPs为$O(B_r B_c d)$。

上述计算循环的总次数为$T_c T_r = \left\lceil \frac{N}{B_c} \right\rceil \left\lceil \frac{N}{B_r} \right\rceil$，因此总的FLOPs为：

$$O\left(\frac{N^2}{B_c B_r} B_r B_c d\right) = O\left(N^2 d\right)$$

Theorem 2. 如果SRAM的size M满足 $d \leq M \leq Nd$。标准Attention对HBM访问的次数为 $\Omega\left(Nd + N^2\right)$，而FlashAttention对HBM访问的次数为 $O(N^2d^2M^{-1})$。

- Theorem 2的证明过程如下。
  需要从HBM读取的数据有：
- Algorithm 1第6行：每次循环读取的$\mathbf{K}_j, \mathbf{V}_j$的size复杂度都为$\Theta(M)$，总size为$\Theta(Nd)$。
- Algorithm 1第8行：每次循环读取的$\mathbf{Q}_i, \mathbf{O}_i$的size复杂度都为$\Theta(Nd)$，总次数为$T_c = [\frac{N}{B_c}] = \Theta(\frac{Nd}{M})$。

FlashAttention对HBM总访问次数的复杂度为：

$$\Theta(Nd + NdT_c) = \Theta(NdTc) = \Theta(N^2d^2M^{-1})$$

---

and CNNs, ISSCC 2024

- MIM4DD: Mutual Information Maximization for Dataset Distillation, NeuIPS 2023.
- RPTQ: Reorder-based Post-training Quantization for Large Language Models. arXiv preprint 2023.
- Post-training Quantization on Diffusion Models. CVPR 2023
- PD-Quant: Post-Training Quantization based on Prediction Difference Metric. CVPR 2023.
- Latency-aware Spatial-wise Dynamic Networks, NeurIPS 2022.
- Flatfish: a Reinforcement Learning Approach for Application-Aware Address Mapping. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2022.
- PTQ4ViT: Post-Training Quantization Framework for Vision Transformers. European Conference on Computer Vision (ECCV), 2022.
- 3DPEE: 3D Point Positional Encoding for Multi-Camera 3D Object Detection Transformers. ICCV 2023.

👋 后摩智能于2020年在南京成立，是国内首家基于 “存算一体” 技术的智能驾驶芯片高新技术企业，在北京、上海、深圳等地方建有研发中心。后摩智能致力于突破智能计算芯片性能及功耗瓶颈，加速人工智能普惠落地。其提供的大算力、低功耗的高能效比芯片及解决方案，可应用于智能驾驶、泛机器人等边缘端，以及云端推理场景。2023年5月，发布了首款基于SRAM的存算一体大算力AI芯片产品，算力高达256Tops。