OliVe：通过硬件友好的离群值-受害者对量化加速大语言模型

Cong Guo*
guocong@sjtu.edu.cn 上海交通大学 上海启智研究院 中国上海

唐嘉明*sakits_
tjm@sjtu.edu.cn 上海交通大学 上海启智研究院 中国上海

胡伟明†
huweim1120@gmail.com 上海交通大学 上海启智研究院 中国上海

Jingwen Leng‡
leng-jw@cs.sjtu.edu.cn 上海交通大学 上海启智研究院 中国上海

Chen Zhang
chzhang1990@gmail.com 微软研究院 中国北京

杨帆
fanyang@microsoft.com 微软研究院 中国北京

刘云欣
li
uyunxin@air.tsinghua.edu.cn 人工智能产业研究院（AIR）、清华大学 中国北京 上海人工智能实验室 中国上海

Minyi Guo‡
guo-my@cs.sjtu.edu.cn 上海交通大学 上海启智研究院 中国上海

朱宇豪
yzhu@rochester.edu 罗切斯特大学 美国纽约州罗切斯特

arXiv:2304.07493v1 [cs.AR] 15 Apr 2023

摘要

随着模型规模不断增大，基于 Transformer 的大语言模型（LLM）取得了巨大成功。大语言模型的规模每两年增长240×，其增速超过了硬件发展，使模型推理的成本日益高昂。模型量化是缓解大语言模型规模与硬件容量之间不断扩大的差距的一种有前景的方法。然而，大语言模型中离群值（即幅值显著的数值）的存在，使现有量化方法的效果大打折扣。以往的离群值感知量化方案采用稀疏性编码技术，将离群值与正态值分离，这一过程需要全局协调（例如，全局稀疏性协调列表）。这会带来复杂的编码/解码硬件逻辑，并需要额外的编排控制器来处理离群值与正常值之间的计算。因此，这种方法缺乏硬件效率，只能获得次优的量化收益。

我们提出了 OliVe，这是一种算法与架构协同设计的解决方案，采用离群值-受害值对（OVP）量化，在本地处理离群值，同时实现较低的硬件开销和显著的性能提升。OliVe 的关键洞见在于，离群值很重要，而其旁边的正常值则不重要。因此，可以牺牲这些正常值（称为受害值）来容纳离群值。由此，OliVe 实现了内存对齐的 OVP 编码方案，可高效集成到现有硬件加速器中，例如脉动阵列和张量核心。因此，与现有的离群值感知加速器 GOBO 相比，基于 OliVe的加速器分别实现了 4.5× 的加速比和 4.0× 的能耗降低，同时模型精度更高。

CCS 概念

• **计算机系统组织** → **神经网络**；**数据流架构**；**单指令多数据**；**脉动阵列**。

关键词

大语言模型、离群值-受害值对、量化

ACM 参考格式：
Cong Guo、唐嘉明、胡伟明、Jingwen Leng、Chen Zhang、杨帆、刘云欣、Minyi Guo 和朱宇豪。2023。OliVe：通过硬件友好的离群值-受害值对量化加速大语言模型。载于《第*50*届计算机体系结构国际研讨会论文集》（*ISCA '23*），*2023*年*6*月*17*日至*21*日，美国佛罗里达州奥兰多。ACM，纽约州纽约市，美国，15页。https://doi.org/10.1145/3579371.3589038

*共同完成本文。†相关工作完成时，作者隶属于上海科技大学。‡Jingwen Leng 和 Minyi Guo 是本文的通讯作者。

允许免费制作本文全部或部分内容的数字版或纸质副本，供个人或课堂使用，但不得以营利或商业利益为目的复制或传播，且副本首页须保留本声明和完整引文。必须尊重非作者拥有的本文组件的版权。允许注明出处后摘录。若要以其他方式复制、重新出版、发布到服务器或分发至列表，则须事先获得特别许可和/或支付费用。请通过permissions@acm.org 申请许可。*ISCA '23*，*2023*年*6*月*17*日至*21*日，美国佛罗里达州奥兰多© 2023 版权所有归所有者/作者所有。出版权已授权给 ACM。ACM ISBN 979-8-4007-0095-8/23/06...$15.00 https://doi.org/10.1145/3579371.3589038

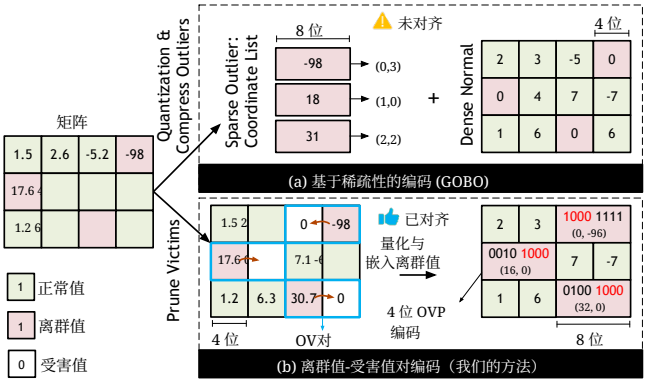


图 1：离群值感知编码对比。(a) 先前的量化工作采用基于稀疏性的编码，将正态数值与离群值分别存储。(b) 我们提出的离群值-受害者对编码在本地存储正态数值和离群值。

1 引言

基于 Transformer 的大语言模型（LLM） [77] 在过去几年中取得了巨大成功。这种成功往往得益于不断增大的模型规模：模型规模每两年增长 240×，显著超过硬件发展速度（每两年3.1×） [24]。因此，大语言模型的推理变得困难且成本高昂。例如，OPT-175B [90],这一近期的基于 Transformer 的大语言模型拥有1750亿个参数，无法装入内存为80GB的最新高端 H100 GPU。

量化 [6, 7, 21, 22, 72, 74, 79, 93] 是降低大型模型推理成本的最高效硬件方式之一。它使用低精度数据类型压缩模型，并借助实用的硬件实现加速计算，例如 TPU [42]和 GPU 张量核心 [60]。

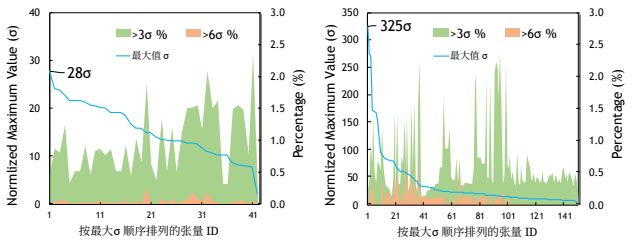
然而，现有的量化方案 [18, 74, 86] 在基于 Transformer 的大语言模型中效果较差。近期研究表明，当模型规模超过某个阈值（例如 60 亿）时，模型性能容易受到仅占极小比例（< 0.1%）的离群值影响，这些离群值的数值远大于正常值 [18]。不加区分地对离群值和正常值进行裁剪，会导致模型准确率显著下降 [18, 82]。因此，与卷积网络（CNNs）相比，业界通常采用更大的位宽（例如 8 位或 16 位）来量化基于 Transformer 的模型。

研究人员提出了各种量化与架构协同设计工作 [39, 61, 75, 82, 85]，以处理 Transformer 模型中的离群值。例如，离群值抑制 [82] 提出抑制离群值。但在较低位宽（4 位）下，它仍会造成显著的准确率损失，这表明要适应离群值的影响并非易事。此外，架构研究人员设计了复杂的离群值感知硬件架构，以高精度存储离群值，从而维持模型精度。这些离群值感知量化框架将张量划分为正常值和离群值，并采用不同方式分别对其编码。对于正常值，采用低精度（例如 4 位）量化的稠密矩阵。而稀疏的高精度数值（例如，8 位和 16 位）离群值可以通过基于稀疏性的编码进行压缩。但遗憾的是，这种编码会导致非对齐内存访问。例如，GOBO [85] 和 OLAccel [61] 使用坐标列表标示矩阵中每个离群值的位置，如图 1a 所示。BiScaled-DNN [39] 利用块稀疏索引格式存储离群值索引，而 DRQ [75] 则使用直接位图表示离群值。这些离群值感知解决方案需要复杂的架构设计，并产生显著的硬件开销来处理离群值。此外，由于内存访问随机且未对齐，基于稀疏性的编码与现有加速器（如图形处理器和 TPU）的内存子系统不兼容。具体而言，GOBO [85] 只能在片外 DRAM 上对权重张量进行解压缩或压缩；它仍依赖图形处理器原有的片上内存和采用高精度 FP16/32 的计算架构。

上述离群值感知架构以一种全局方式将正常值与离群值分离。例如，GOBO [85]在量化和计算过程中使用全局稀疏坐标列表，从而带来较大的硬件开销，且性能收益较低。在本工作中，我们旨在设计一种以局部方式处理离群值且具有高硬件效率的架构。为此，我们将张量中两个连续的固定大小数值分为一组，并分析它们对模型准确率的影响。这里有三种对：i）由两个正常值组成的正常对；ii）由一个正常值和一个离群值组成的单离群值对；iii）由两个离群值组成的双离群值对。我们观察到，在训练良好的大语言模型中，第三种双离群值对几乎从未出现。对于第二种单离群值对，我们发现，仅保留其中的离群值并剪枝其正常值（即将其视为零）就足以维持模型准确率。

基于上述观察，我们提出了一种新颖的离群值感知量化架构，称为 OliVe，它基于离群值-受害者配对（OVP）编码。OliVe 的显著特点是与内存对齐，因此具有硬件友好性。如图1b所示，OliVe 首先将与离群值相邻的正常值剪枝为零。这些被剪枝的正常值称为**牺牲值**，它们牺牲自身为离群值腾出空间。随后，我们利用牺牲值提供的额外空间，将离群值嵌入低精度矩阵中。

由于以下原因，OliVe 能够在硬件开销较低的情况下，为大型 Transformer 模型保持较高的准确率。首先，OliVe 引入牺牲值来处理大语言模型中的离群值。牺牲值的作用类似于模型剪枝 [36]。尽管裁剪少量（0.1%）离群值会导致准确率灾难性下降 [18, 82]，剪除相同比例的“正常”数值对模型准确率的影响却很小（下降 < 0.1%）。因此，OliVe 将这些无关紧要的数值作为离群值的牺牲值（“剪除”），从而能够采用更激进的编码方案来容纳极其重要的数值。其次，OVP 编码遵循特定的离群值-受害值（或受害值-离群值）模式，以较低的硬件开销实现内存对齐。每个牺牲值都与一个离群值相邻，且离群值-受害值对必须与内存访问模式对齐。例如，在图1b中，OV对中的右侧离群值 −98 需要一个左侧牺牲值，而左侧离群值 17.6 和 30.7 则需要右侧牺牲值。这样就能高效地对齐 8 位（1 字节）内存访问。这种设计实现了完全局部化的离群值解码/编码过程。



(a) ImageNet 上的 ResNet-18。

(b) MNLI 上的 BERT$_{base}$

图2：卷积神经网络模型与 Transformer 模型的离群值比较。σ 表示张量的标准差。我们将最大值除以 σ 进行归一化，以绘制最大值 σ 曲线（左侧 y 轴）。$> 3\sigma\%$ 和 $> 6\sigma\%$（右侧 y 轴）分别表示 $> 3\sigma$ 和 $> 6\sigma$中数值所占的百分比。

OliVe 对离群值和正常值采用不同的数据类型，因为它们的动态范围和表示格式各不相同，例如 int4 和 FP4。如图1b所示，我们为 4 位 OV对提出了一种新颖的编码方法（第 3 节）：将一个 4 位离群值和一个 4 位牺牲值组合成一种特殊的 8 位格式，与原始的 int8 或 FP8 不同。得益于其硬件友好且兼容的设计，OliVe 可以轻松集成到现有的量化框架和加速器架构中，例如 Google TPU 中的脉动阵列 [41]以及 NVIDIA GPU 中的张量核心 [58, 60]。OliVe 还能够原生支持混合精度和混合类型架构，展现出其在更大规模 Transformer 模型中的灵活性与实用性。

据我们所知，OliVe 是首个将 Transformer 训练后量化（PTQ）[4],推至 4 位水平的工作；该方法在量化后无需重新训练，且权重和激活张量的准确率损失仅为 < 1%。令人惊讶的是，OliVe 对 BERT [19]和 BART [49] 模型进行 4 位 PTQ 所取得的准确率，优于离群值抑制 [82], 这一最先进 Transformer 量化方法的 6 位 PTQ 结果。与现有的离群值感知加速器 OLAccel [61] 和 GOBO [85] 相比，基于 OliVe 的加速器分别实现了 3.8× 和 4.5× 的性能提升，以及 2.1× 和 4.0× 的能耗降低。更重要的是，与其他针对离群值设计的架构相比，基于 OliVe 的加速器具有更全面且更实用的适用性。

本文的贡献如下。

- 我们进行成对重要性分析，发现离群值十分重要，而其相邻的正常值并不重要。这揭示了离群值-受害值对（OVP）的算法机遇：牺牲共置的正常值（称为受害值）来容纳离群值。
- 我们提出了基于OVP的量化框架，称为 OliVe，其中包括高效的硬件编码和新颖的异常值表示数据类型。
- 我们提出了OliVe 量化的高效架构实现与集成，并表明其效率和优势优于现有的异常值感知量化算法和硬件加速器。

2 动机：对齐离群值

在本节中，我们首先说明，与卷积神经网络（CNN）相比，Transformer模型中的离群值更加显著且重要。以往的研究 [74, 75, 85, 86] 提出了采用自适应位长的异常值感知量化微架构，以实现低比特量化，但需要大量硬件资源来处理可变长度数据，从而导致非对齐内存访问，并且与现有加速器（例如图形处理器）的内存子系统不兼容 [60]。相比之下，我们提出了一种内存对齐且硬件友好的方法，称为异常值-牺牲值对机制。该方法受深度神经网络剪枝以及我们针对Transformer的异常值组位置分析启发。我们可以剪枝一些“牺牲值”，从而腾出空间，将高精度异常值嵌入内存对齐低比特张量中，同时几乎不会造成准确率损失。

2.1 离群值至关重要

我们在图 2 中直观展示了 Transformer 的离群值有多么重要。我们采用正态分布的经验 3σ **规则** [83]，将数值划分为离群值和正常值。我们选用 ResNet-18 [37] 作为卷积神经网络模型的代表，选用 BERT$_{base}$ [19] 作为 Transformer 模型的代表。我们使用正态分布拟合深度神经网络张量，即公式 1，其中 x为数值，μ为均值，σ为标准差。我们将张量转换为标准正态分布。

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{1}$$

我们收集所有张量的最大值，并将其按 σ（最大值 σ）归一化。我们按照张量的最大值 σ 排序并绘制图表，如图 2 所示。

大多数张量都符合正态分布 3σ 规律，即约 99.7% 的数值位于均值的三个标准差范围内。大多数张量的离群值（$> 3\sigma$）占比低于 0.5%，且张量中的 $> 6\sigma$ 数值极少。因此，正常值相对集中，这表明我们可以用较窄的范围量化正常值，以提高量化分辨率。

更明显的观察结果是，Transformer 的最大值 σ 比卷积神经网络高一个数量级。一些研究 [14, 43] 表明，尽管卷积神经网络模型中的离群值被裁剪，在超低位精度（例如 4 位）下，仍可通过重新训练算法将准确率恢复到原始水平。然而，对于离群值显著得多的 Transformer模型，这种做法颇具挑战。最先进的量化研究 [18, 82] 也得出了类似的观察结果，并指出，由于离群值的存在，大规模 Transformer模型只有采用高精度量化才能达到原始准确率。因此，不裁剪离群值并予以保留，将显著有利于 Transformer模型的量化。

2.2 离群值未对齐

离群值的重要性引起了众多研究者的关注，并催生了多种离群值感知架构，如表1所示。OLAccel [61] 和 GOBO [85] 采用类似的方法，利用坐标列表标示离群值的位置，并使用高精度（8 位或 16 位）量化。BiScaled-DNN [39] 和 DRQ [75] 则分别采用块稀疏索引和位图。

加速器	编码	已对齐内存?	GPU兼容吗?
OLAccel [61]	坐标列表	No	No
BiScaled-深度神经网络 [39]	块稀疏索引	对齐数据 未对齐索引	No
DRQ [75]	二进制掩码映射	未对齐数据 对齐索引	No
GOBO [85]	坐标列表	No	仅动态随机存取存储器
OliVe（我们的方法）	**离群值-受害值对**	**Yes**	**Yes**

表1：现有的离群值感知加速器与我们提出的 OliVe 方法的比较。

BiScaled-DNN 使用相同的位宽量化所有数值，但对正常值和离群值采用不同的缩放因子，且这些数值已对齐。然而，块稀疏方法中额外压缩的索引未对齐。相反，DRQ 的位图已对齐，但数据以混合方式存储，因此 4 位和 8 位数值未对齐。

总之，已有研究基于离群值的稀疏性设计离群值感知架构，这会导致非对齐内存存储和访问。更严重的是，基于稀疏性的编码索引与离群值彼此分离。因此，它们需要额外的离群值控制器来解析离群值的索引，并协调正常值与离群值之间的计算。例如，GOBO 和 OLAccel 的额外离群值控制器开销分别高达处理单元（PE）阵列总面积的 55% 和 71% [61, 85]。针对离群值的基于稀疏性的编码也与现有加速器的内存子系统**不兼容**。对于 GOBO 设计 [85]，，它只能在图形处理器的动态随机存取存储器层级对内存进行压缩和解压缩。这极大地限制了其所提出的离群值感知架构的适用性。

因此，应提出一种更加硬件友好且适用的离群值解码/编码方法，以适配离群值感知量化。我们提出的 OliVe架构能够实现内存访问对齐，并且基于 OVP机制兼容现有加速器。

2.3 离群值与受害者分析

通常，借鉴深度神经网络剪枝的基于稀疏性的编码，是处理稀疏离群值的一种直接且有效的方案。然而，这些工作忽略了量化与剪枝之间的差异。对于剪枝，被剪除的零值不会参与计算。因此，剪枝方法必须压缩

配对类型	正常值-正常值	离群值-正常值	离群值-离群值
BERT$_{base}$ [19]	99.12%	0.84%	0.04%
BERT$_{large}$ [19]	99.24%	0.71%	0.05%
GPT2-XL [66]	98.80%	1.14%	0.06%
OPT-6.7B [90]	99.33%	0.64%	0.03%

表2：三种配对类型的百分比。

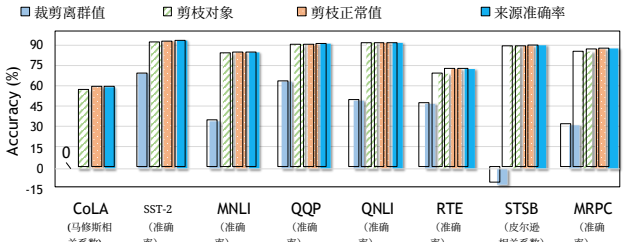


图3：多种剪枝方法的准确率对比。

稀疏值，并采用基于稀疏性的编码。对于量化，量化后的正常值占绝大多数，而且需要参与计算。因此，离群值可以利用正常值实现内存对齐，而不必采用基于稀疏性的编码。

如第1节的图1b所示，我们借鉴了剪枝的思路，但采用了与以往工作不同的视角。新方法采用**离群值-受害值配对**（OVP）机制。我们首先剪去一些经过量化的低精度正常值，并将其称为**牺牲值**。这些牺牲值与离群值相邻，为高精度异常值腾出额外空间。因此，我们可以将离群值嵌入其原始位置，而无需显式使用稀疏索引。这既能避免复杂的索引硬件，也能使其兼容图形处理器。为使内存对齐，我们根据离群值在配对中的位置，将其区分为“右侧离群值”和“左侧离群值”。我们为左侧离群值分配一个右侧牺牲值（例如图1b中的17.6），为右侧离群值分配一个左侧牺牲值（例如图1b中的 −98）。

OVP机制基于我们对大型Transformer模型的观察，其中包括 BERT-base [19], BERT-large [19], GPT2- XL [66], 和OPT-6.7B [90]。我们收集所有张量，计算其标准差 σ，并依据 3σ规则将数值分为正常值（$< 3\sigma$）和离群值（$> 3\sigma$）。随后，我们将每两个相邻数值配成一对（不重叠），由此得到三种类型：正常值-正常值对、离群值-正常值配对和离群值-离群值对，如表2所示。这三种类型分别包含两个正常值、一个正常值和一个离群值，以及两个离群值。

表2显示，大多数（约99%）配对都是正常值-正常值配对，只有约1%是离群值-正常值配对。对于离群值-离群值对，需要剪除其中较小的离群值。幸运的是，在所有研究的模型中，离群值-离群值对出现的概率极低，不到0.06%。因此，离群值分布极为分散，我们可以保留大多数离群值。

我们还在 GLUE 数据集 [78], 上，对 BERT$_{base}$ 模型 [82] 进行了准确率实验，如图3所示。首先，我们将离群值裁剪到 3σ，裁剪是量化中常用的方法。随后，我们将牺牲值和正常值剪除为零。牺牲值与离群值相邻，而正常值则按照与离群值相同的数量随机剪除

其余数值均保留为全精度（单精度浮点数）。尽管如图 3 所示，裁剪离群值时仅裁剪了少量离群值（约 1%），但对于 BERT 模型而言，准确率损失仍不可接受。结果强调了离群值在基于Transformer的模型中的重要性。作为对比，剪枝随机正常值几乎不会造成准确率损失，与来源准确率相比几乎没有差异。仅剪枝受害值导致的准确率下降可以忽略不计，相较于剪枝正常值而言，这是因为受害值中包含一些离群值，这些离群值是由于

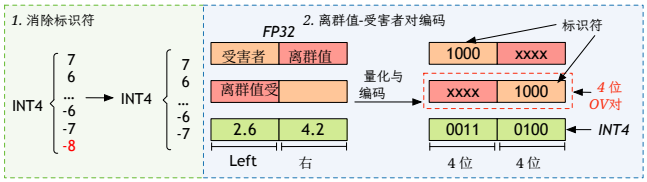


图4：4 位离群值-受害者对编码。

离群值-离群值配对而产生的，并且其位置与相邻离群值相对应。

总之，我们的分析表明，离群值很重要，而受害值则不然，因此我们可以牺牲受害值来容纳离群值。这促使我们设计硬件友好的 OVP 机制，提供已对齐的离群值感知量化，以加速大型 Transformer 模型。下一节将介绍离群值-受害者对编码设计。

3 离群值-受害者对编码

本节介绍离群值-牺牲值对（OVP）编码的具体细节。对于离群值和正常值，该编码在全局上保持一致，同时在局部可加以区分。OVP 编码能够维持全局对齐的内存访问，并以可忽略的开销在局部区分离群值。对于正常值，我们可以支持多种数据类型，以适配自适应数据类型。对于离群值的编码，我们设计了一种异常值专用数据类型——自适应偏置浮点数（abfloat），它可以避免正常值与离群值之间的范围重叠，从而提高离群值编码中数值表示空间的利用率。最后，基于 OVP 编码，我们提出了一种框架，可自动选择 OVP 编码的离群值阈值，以确定合适的离群值-受害值对比例。

3.1 OVP 编码算法

基于前面对逐对张量值的分析，共有三种配对类型：正常值-正常值、离群值-正常值和离群值-离群值。对于离群值-正常值配对，将剪除配对中的正常值，并将其转变为受害者。对于离群值-离群值配对，则保留较大的值并剪除另一个值。这样，我们便能在深度神经网络张量中得到正常值-正常值对和异常值-受害者对。

离群值标识符。为了与正常值-正常值配对区分开来，我们需要为离群值-受害值对设置一个特殊标识符。这个独特的标识符不能出现在正常值-正常值配对中，这意味着我们需要从正常值的表示中去掉一个数值。例如，如图 4 所示，我们采用有符号 int4（4 位整数）进行正常值量化。原始 int4 可以表示范围为 $[-8, 7]$ 的整数，其中 1000_2 表示 -8 的值。首先，我们将 1000_2 设为离群值标识符，并从 int4 中移除数值 1000_2，使其编码范围变为 $[-7, 7]$。其次，我们采用 4 位 OVP 编码对离群值-受害者对进行量化。我们将受害者设为离群值标识符 1000_2，并使用离群值专用数据类型对离群值进行量化（第 3.3 节）。自然地，OVP 对有两种类型，即左侧离群值（O-V）对和右侧离群值（V-O）对。借助独特的离群值标识符设计，我们无需额外的索引位即可隐式区分它们（第 4.2 节）。

算法 1： 4 位 OVP 编码算法。

输入： 数值，val_1，val_2；离群值阈值，T。
输出： OVP 编码，out_1，out_2。

```
def OVPairEncoding(val1, val2, T):
    如果 val1 > T 且 val1 > val2 则
        out1 =异常值量化(val1);
        out2 = 1000_2; // 离群值标识符。
    否则，如果 val2 > T 则
        out1 = 1000_2
        out2 =异常值量化(val2);
    else
        out1 =正常值量化(val1);
        out2 =正常值量化(val2);
    返回 out1, out2
```

算法 1 展示了 4 位 OVP 编码算法，该算法需要同时读取两个数值，而这一要求很容易满足。对于硬件实现，我们可以为编码器添加一个缓冲区。此外，OVP 编码器可以嵌入量化单元中实现，且开销可以忽略不计。对于软件实现，我们可以让一个线程同时处理两个数值。因此，该编码算法可以在硬件和软件中高效实现，详情将在后文介绍。

3.2 正常值的数据类型

对于正常值，我们基于先前的工作 [32],，该工作支持多种数据类型，包括 int4、flint4（4 位浮点数）和 int8，如表 3 所示。int4 是 4 位整数值量化中最广泛使用的数据类型之一，其数值范围为 $[-7, 7]$。flint4 数据类型由先前的工作 ANT [32]，提出；该工作表明，根据张量的分布选择数据类型，可以实现最先进的性能和准确率。

基于上述洞见，我们在 OVP 对编码中也采用混合数据类型来量化正常值。对于 flint4，我们使用与离群值标识符相同的二进制值 1000_2。具体而言，flint4 的 1000_2 对应于 -0，而原始设计中并未使用该值。换言之，我们的 OVP 编码可无缝适用于 flint4，不会浪费任何数值表示。我们使用原始的 flint4 编码算法 [32] 来量化正常值。

此外，OVP 编码通常还可以扩展到更高精度的量化，例如 8 位量化。同样，8 位正常值

数据类型	数值	离群值标识符
int4	$0, \pm1, \pm2, \pm3, \pm4, \pm5, \pm6, \pm7$	1000_2 (-8)
弗林特4 [32]	$0, \pm1, \pm2, \pm3, \pm4, \pm6, \pm8, \pm16$	1000_2 (-0)
int8	$0, \pm1, \pm2, \cdots \pm126, \pm127$	10000000_2 (-128)

表 3：OVP 编码正常值的数据类型。

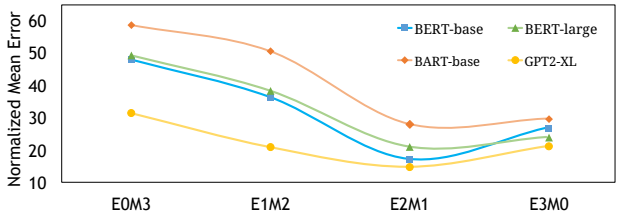


图 5：采用不同数据类型量化最大离群值时的舍入误差。实验在 BERT-base、BERT-large、BART-base 和 GPT2-XL 上进行。

也需要排除一个数量。例如，int8 可以表示 $[-128, 127]$ 个整数，我们可以将 10000000_2 设为 int8 的离群值标识符，并将其范围缩小到 $[-127, 127]$。同样，编码算法可以轻松扩展为同时读取两个 8位元素。

3.3 离群值的数据类型：abfloat

接下来，我们使用异常值专用数据类型量化离群值。较大的离群值通常取值范围较广，因此我们使用基于浮点数的数据进行量化。我们提出了一种称为**自适应偏置浮点**的数据类型，简称 abfloat。其核心思想是为指数添加适当的偏置，使所有编码值都能跳过正常值所在的区间，从而为离群值提供更大的取值范围。

浮点到定点转换。 为了容纳正常值并避免出现小数，我们首先将浮点编码转换为带指数的定点数。此外，定点数适用于硬件实现，且开销低于浮点数。我们通过以下公式将浮点数转换为定点数，

$$\texttt{sign} \times (1 \ll \texttt{mb} + \texttt{mantissa}) \ll (\texttt{exponent} + \texttt{bias}), \quad (2)$$

其中 mb 为尾数位宽。因此，这种定点编码方案对硬件实现更加友好、高效，因为它只涉及移位操作。表 4 展示了定点 E2M1 数据类型的示例。

自适应偏置。 显然，表 3 和表 4 表明，定点 abfloat 的取值范围与正常值重叠。例如，int4 和 E2M1 包含相同的数值：3、4 和 6。另一个例子是，除了 24 以外，flint4 和 E2M1 的数值范围几乎相同。因此，我们需要使用自适应偏置来调整 abfloat 的取值范围。例如，我们为 E2M1 设置偏置 $= 2$，其真实值将扩展至 $\{12, \cdots, 96\}$，从而与 int4 的正常值互补。类似地，我们设置偏置 $= 3$

二进制	指数 整数		实数值
000	0	0	0
001	0	3	$3 \times 2^0 = 3$
01x	1	2, 3	$2 \times 2^1 = 4, 3 \times 2^1 = 6$
10x	2	2, 3	$2 \times 2^2 = 8, 3 \times 2^2 = 12$
11x	3	2, 3	$2 \times 2^3 = 16, 3 \times 2^3 = 24$

表 4：3 位无符号 E2M1，表示指数占两位、尾数占一位，偏置为 $= 0$。

算法 2： abfloat 编码算法。

输入： 元素 e；偏置， b；
输出： 量化元素 q；

```
def AbfloatQuant(e, b):
    // 获取指数和基整数。
    exp = ⌊log2(abs(e))⌋ − 1;
    base_int = Round[e/2^exp];
    如果 base_int == 4那么
        exp = exp + 1;
        base_int = base_int − 2;
    // 编码为 abfloat 数据类型。
    exp = exp − b;
    base_int = base_int & 1;
    unsigned_q = Concat(exp, base_int);
    q = Concat(e < 0, unsigned_q)
    返回 q
```

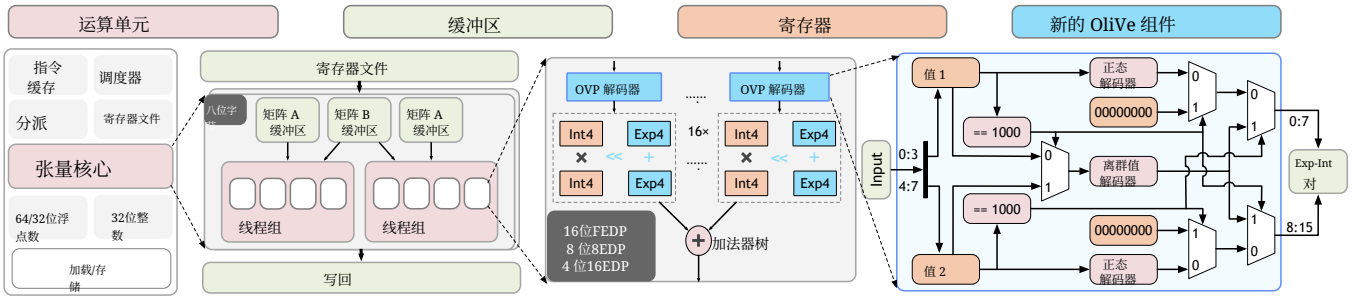
并将 flint4 数据类型的取值范围扩展至 $\{24, \cdots, 192\}$。我们设计了一种新的解码器和指令，以在加速器中为 abfloat 实现自适应偏置（第 4.2 节）。

E2M1 Abfloat。4 位有符号浮点数的指数和尾数有四种可能的配置：E0M3、E1M2、E2M1 和 E3M0。它们的范围和精度各不相同。我们开展了以下实验，以选择最合适的配置作为最终的离群值专用数据类型。为了适应离群值的广泛取值范围，我们使用所有 abfloat 类型对 Transformer模型中的最大离群值（即图 2 中的 Max σ）进行量化。随后，我们收集了平均绝对误差，如图 5 所示。我们发现，在所有测试中，E2M1 的误差都最小；它既提供了足够大的范围，也具备一定的精度，并且在后续评估中表现最佳。同样，对于 8 位 abfloat，我们采用有符号E4M3。

算法2详细展示了如何将一个元素编码为 abfloat。离群值编码是一种逐元素函数，可以在软件和硬件上高效实现。离群值编码还应排除离群值标识符，否则解码器无法区分离群值-受害值对。abfloat 有两个零值：1000（-0）和 0000（0）。因此，我们禁止离群值使用 1000 和 0000，以避免与离群值标识符冲突。

3.4 Quantization Framework

现在，我们采用 OVP（离群值-受害值对）编码来量化 Transformer模型。为确定缩放因子（即离群值-受害值阈值），我们将 OVP 编码融入现有的均方误差（MSE）最小化算法中；许多量化研究都普遍采用该算法 [4, 6, 88]。基于 OVP 的量化算法会确定区分离群值与正常值的阈值。一方面，较小的阈值会产生更多离群值-受害值对，有望将量化误差（即 MSE）降至最低。另一方面，这也会提高离群值-离群值对的比例，即配对中的两个数值都是离群值。如果这类离群值-离群值对过多，剪枝离群值会导致 MSE 增大。因此，为了提高准确率，我们需要控制离群值-离群值对的比例。



(a) GPU 张量核心　　(b) 离群值-受害者对解码器

图 6：OliVe　　在 GPU 张量核心上的集成 (a)，只需一组轻量级的 OVP 解码器 (b)

在我们的工作中，我们关注训练后量化（PTQ）[57]，这种方法无需重新训练，因此最适合大型模型，因为训练这些模型的成本很高。不过，我们仍需使用一批来自**训练集**的数据来选择缩放因子。直观来说，受 3σ 规则启发，我们将 3σ 作为初始缩放因子。随后，算法会在该基准值的特定范围内搜索 MSE 最小的最佳缩放因子；我们的评估表明，这种方法效果良好。对于量化感知训练（QAT）[57]，我们可以通过使用直通估计器（STE）[5]重新训练模型来获得合适的缩放因子。

4 OliVe架构

本节介绍如何将 OliVe 集成到图形处理器和输出驻留脉动阵列架构中。随后，我们将介绍用于上述离群值-受害者对编码和离群值数据类型的硬件解码器。在这些架构上，我们提出的 OliVe 架构可直接支持混合精度 [60, 72] 和混合数据类型 [60, 72]，从而高效地量化重要性和分布各不相同的深度神经网络张量。

4.1 GPU张量核心

我们首先在图6a中介绍如何将 OliVe设计集成到图形处理器的张量核心架构中。我们采用图灵架构 [59] 作为基准图形处理器，它有 68 个流式多处理器（SM），每个 SM 有 8 个张量核心（共 544 个），如表5所示。根据先前工作的建模 [67]，每个张量核心有两个八元组，每个八元组包含 8 个四元素点积（FEDP）。因此，有

$$68 \times 8 \times 2 \times 8 \times 4 = 34\ 816,$$

位浮点乘法器。图灵架构原本支持混合精度计算。例如，采用图灵架构的 RTX 2080Ti GPU [59] 在 16位浮点数、8位整数和 4位整数下，分别可提供 107.6、215.2 和 430.3 每秒万亿次运算（TOPS）。因此，我们假设张量核心可以同时支持 8位8EDP（八元素点积）和 4位 16EDP（16 元素点积），如图6a所示。

架构	SM	TC	16 位单元	8 位单元	4 位单元
图灵 [59]	68	544	34,816	69,632	139,264

表5：Turing GPU 架构。

我们可以轻松地将提出的 OliVe架构嵌入采用 SIMD 架构的图形处理器中。首先，我们为每个 16EDP 配置 4 位离群值－受害者配对解码器（图6b）。为支持新的 OliVe 数据类型，我们为每个 16EDP 添加一个加法器和一个移位器。同样，我们也为 8EDP 单元设计了 8 位解码器。

4.2 解码器

离群值-受害值对解码器。为支持离群值-受害值对解码，我们设计了一种可轻松嵌入现有加速器的新型解码器。如图6b所示，该解码器读取1字节数据（这是许多架构中最小的可寻址存储单元）以及恰好一对数值。随后，解码器将离群值标识符 1000_2 转换为0，并使用离群值解码器对离群值进行解码。为适应离群值 abfloat数值的计算，该解码器将生成一个指数-整数对。因此，对于常规 int4数据类型，解码器需要附加一个 0000_2 作为指数数量。对于 flint4，我们利用其原有解码器 [32] 来获取指数-整数对。

离群值解码器。上述 OVP 解码器包含一个用于 E2M1 abfloat 数据类型离群值的离群值解码器。图7展示了 4 位 abfloat 解码器的设计细节。对于一个 4 位 E2M1 abfloat 数值 $x = (b_2 b_1 b_0)_2$，以下公式用于解码指数和整数：

$$\texttt{exponent} = \texttt{bias} + (b_2 b_1)_2$$

$$\texttt{integer} = \begin{cases} 0 & if\ x = 000_2 \\ (1 b_0)_2 & otherwise \end{cases}$$

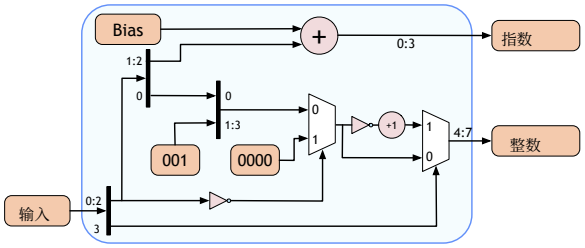


图 7：用于离群值的 4位abfloat解码器。

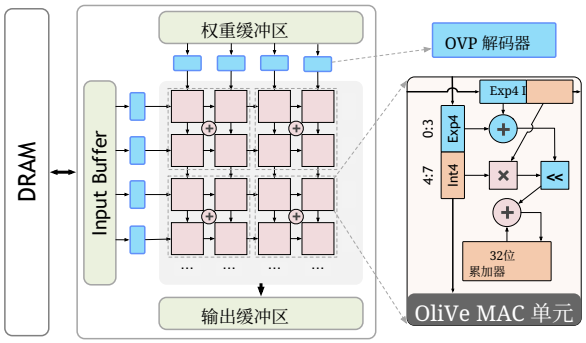


图 8：OliVe 在脉动阵列上的集成。

例如，当偏置为 2 时，一个数值 0101_2 是 48_{10}，，因为它的指数为 $2_{10} + 10_2 = 4_{10}$，基整数为 $11_2 = 3_{10}$。因此，它的实数值为 $3 \ll 4 = 48$。

同样，我们还设计并实现了 8 位离群值-受害者对解码器和 E4M3 abfloat 离群值解码器，它们是 4 位实例的直接扩展。因此，由于篇幅有限，我们不再介绍其细节。

4.3 脉动阵列

脉动阵列（SA）的集成方式如图8所示。SA采用与图形处理器相同的离群值-受害者对解码器设计（图6b），这体现了我们设计的广泛适用性。不过，与图形处理器不同，我们只沿边界布置解码器，因此可以省去大部分解码器。例如，如果阵列大小为 $n \times m$，我们只需要 $n+m$ 个解码器，而不是 $n \times m$ 个。这是 SA 相较于图形处理器 SIMD 架构的一项优势。我们提出的基于 OliVe 的数据类型还可以通过额外的加法器和移位器支持脉动阵列处理单元（PE）。我们每四个 PE 增加一个额外的加法器，以支持高精度量化，例如 int8。

4.4 OliVeMAC 单元

对离群值和正常值解码后，它们都会转换为统一的指数-整数对。为了支持解码后的指数-整数对运算，我们需要为定点乘加（乘法与累加）单元添加一个移位器和一个加法器，如图8以及图6中4位16EDP单元所示。例如，有两个指数-整数对 $< a,\ b >$和 $< c,\ d >$，其中 a和 c是指数，b和 d是整数，而 $< a,\ b >$表示：

$$< a, b >= b \ll a$$

然后，我们可以得到结果：

$$\begin{aligned} & < a, b > \times < c, d > \\ = & (b \times d) \ll (a + c) \\ = & < a + c, b \times d > \end{aligned}$$

请注意，最终结果可以用 32 位整数存储。

4.5 混合精度

如第 3 节所述，OliVe 量化支持对正常值使用 int8，对离群值使用 E4M3 abfloat。因此，我们提出了适用于高精度数据类型的混合精度处理单元（PE）。

8 位整数。 GPU 张量核心架构最初就是为混合精度计算而设计的。对于脉动阵列，我们的架构天然支持使用四个 4 位处理单元进行 8 位计算 [72]。对于一个 int8 数字 x，可以将高 4 位和低 4 位拆分为两个 4 位数 h和 l，并且 x可以表示为：

$$x = (h_x \ll 4) + l_x =< 4, h_x > + < 0, l_x > .$$

然后，我们可以将两个 int8 数字 x和 y相乘：

$$\begin{aligned} x \times y = & \underbrace{< 4, h_x > \times < 4, h_y >}_{PE0} + \underbrace{< 4, h_x > \times < 0, l_y >}_{PE1} \\ & + \underbrace{< 0, l_x > \times < 4, h_y >}_{PE2} + \underbrace{< 0, l_x > \times < 0, l_y >}_{PE3} \end{aligned}$$

因此，我们可以使用四个 4 位处理单元计算上述四次乘法，并将乘积累加，从而得到 $x \times y$ 的最终乘积值。

8 位 `Abfloat` 同样，可以使用相同的方法支持 8 位 abfloat 的乘法。对于一个 8 位 abfloat 数 z，首先将其解码为指数 e_z和整数 i_z。对于 i_z，我们同样将其拆分为

$$i_z = h_z << 4 + l_z \qquad z =< 4 + e_z\ h_z >(), \text{然后,,。} + < e_z\ l_z >$$

因此，可以使用相同的方法，通过四个 4 位处理单元执行 8-位 abfloat 乘法，其中 abfloat 比 int8 多一个 e_z。

在最极端的情况下，两个 abfloat 离群值可能会相乘。由于我们采用 32 位整数作为累加器，最大乘数不应超过 $\sqrt{2^{31} - 1}$。因此，对于 abfloat 类型的离群值，我们会将其绝对值裁剪到 $2^{15} < \sqrt{2^{31} - 1}$ 以内，以避免 int32 累加器溢出。我们的实验表明，Transformer模型中的离群值远小于 2^{15}。具体而言，经过归一化和量化后，2^{15} 约为 768σ。如图 2 所示，离群值的最大值不超过 325σ。因此，我们观察到，实际应用中没有离群值被截断。

4.6 指令集

对于 4 位张量核心，Turing GPU 架构采用指令 mma。s32。s4。s4。s32。这四个操作数是矩阵 D(32位整数)、 A(int4)、 B(int4)和 C(32位整数)，以及 $D = A \times B + C$。为了支持图形处理器上的基于 OVP 的计算，我们设计了一条名为 mmaovp 的新指令：

$$\underbrace{mmaovp}_{\texttt{OVP-MMA}}\texttt{.s32.}\underbrace{ovpi4}_{\texttt{int4}}\texttt{.}\underbrace{ovpf4}_{\texttt{flint4}}\texttt{.s32.}\underbrace{s4}_{\texttt{bias}}\texttt{.}$$

此外，由于数据类型采用内存对齐设计，OliVe 保留了 GPU 的原始编程接口。我们可以将原始的基于 int 的指令替换为基于 OVP 的指令（例如 mmaovp），从而轻松构建支持 OVP 的 DNN 量化框架。因此，我们的 OliVe 框架具有广泛且实用的适用性，这是 OliVe 最显著的优势。

5 评估

在本节中，我们评估采用 OliVe 量化的 LLM 准确率。我们还分别展示 OliVe 在 GPU 和脉动阵列上的面积开销、加速比和能耗效率。

5.1 方法论

框架与评估模型。为评估我们的 OliVe 量化框架，我们使用 PyTorch [62]实现该框架。我们在 GLUE 基准的八个数据集上评估 BERT-base [19],BERT-large [19],和 BART-base [49], 这三种最常用的语言模型。此外，我们在摘要任务 SQuAD v1.1 和 SQuAD v2.0 [68]上评估 BERT-base [19] 和 BART-base [49] 。为了验证我们的量化框架在大语言模型上的效果，我们还在 Wikitext103 [83] 和 C4 [20] 数据集上评估 GPT2-XL [66],BLOOM-7B1 [70], 和 OPT-6.7B [90] 。对于上述所有模型，我们均使用 huggingface 代码库中的最先进检查点 [55]。

量化基线。 我们将 OliVe 与现有量化方法进行比较，包括 GOBO [85], 离群值抑制 [82], Q8BERT [86], 和 ANT [32]。离群值抑制 [82] 是最先进的 Transformer 量化方法。GOBO [85] 也是一种离群值感知量化方法。Q8BERT [86] 是一种将 GEMM 运算量化为 8 位的方法。ANT [32] 是一种硬件友好的量化框架，在性能和准确率方面均取得了最先进的结果。

加速器基线。 我们将 OliVe 与五种深度神经网络量化加速器的性能和能耗进行比较，包括 OLAccel [61], AdaptivFloat [76] （简称 AdaFloat）、GOBO [61], ANT [32], 以及 GPU 中原有的 int8 张量核心 [59]。OLAccel [61]首次提出了面向卷积神经网络的离群值感知量化架构。我们将 OLAccel 扩展至基于 Transformer 的模型，并采用元素级混合精度权重和激活值量化。AdaFloat [76] 通过张量级指数偏置扩展了浮点类型。GOBO [85] 与 OLAccel 类似，但仅支持对基于 Transformer 的网络进行权重量化。

OliVe 实现。我们使用 Verilog RTL 实现解码器，并通过 Synopsys设计编译器 [47] 和台积电 22纳米工艺库对其进行综合，以估算面积、延迟和功耗。我们使用 CACTI [56] 估算片上存储器的面积和功耗。我们将 OliVe 集成到图形处理器和硬件加速器中，以评估端到端性能和能耗。

在图形处理器集成与评估方面，我们基于 NVIDIA 2080 Ti 架构的配置修改并扩展 GPGPU-Sim 4.0 [3] 和 AccelSim [45] 。我们使用 AccelWattch [46], 、 GPUWattch [48], 和 CACTI [56] 进行能耗估算。Transformer层中的大多数操作都是矩阵乘法。对于张量核心上的通用矩阵乘法实现，我们使用 CUTLASS [44], , 这是英伟达的开源实现。

在加速器评估中，我们将 AdaFloat、OLAccel 和 ANT 与 OliVe 进行比较。我们基于 DnnWeaver [71]开发了一个周期级模拟器，用于估算 OliVe 的整体性能。尽管 DnnWeaver [71] 是一个现场可编程门阵列工具集，但此前的深度神经网络量化加速器（包括 BitFusion [72],和 ANT [32],）已扩展其前端，以支持专用集成电路性能和能耗模拟。由于 OliVe 并未重新设计基线加速器

方法	算法	CoLA	SST-2	MNLI	QQP	MRPC
$BERT_{base}$	32位	59.60	93.35	84.94	90.91	87.75
Ours	**4位后训练量化**	**59.30**	**92.43**	**84.10**	**90.36**	**87.99**
ANT	4位量化感知训练	53.91	**92.43**	83.45	-	-
ANT	4位后训练量化	42.90	90.48	73.36	78.04	68.87
OS	4位量化感知训练	50.56	91.86	83.05	90.33	84.31
OS	6位训练后量化	54.40	91.86	82.02	88.94	83.33
Q8	8位量化感知训练	58.48	92.24	-	-	-
$BERT^{large}$	32位	63.35	93.46	86.65	91.07	87.99
Ours	**4位后训练量化**	**63.99**	**92.89**	**84.89**	**90.14**	**86.52**
$BART_{base}$	32位	56.32	93.35	86.45	91.34	87.50
Ours	**4位后训练量化**	**54.30**	**92.89**	**85.33**	**91.23**	86.76
OS	4位量化感知训练	50.83	92.43	84.57	90.93	**87.01**
OS	6位训练后量化	44.51	90.94	82.98	88.45	80.88

表 6：GLUE 数据集上的结果。Q8 和 OS 分别是 Q8BERT [86] 和离群值抑制 [82] 的简称。此前的工作未报告 $BERT^{large}$ 上的结果，因此我们仅与原始全精度模型进行比较。

在架构方面，我们可以直接在模拟器中嵌入新的 OliVe 相关指令和数据格式，而不会破坏原有的模拟流程。换言之，我们使用并修改了 BitFusion [72, 73], 和 ANT [32, 33] 的开源实现。

5.2 准确率结果

我们首先评估 OliVe 量化框架在不同任务和数据集上的准确率，这是将其用于降低大语言模型（LLM）推理成本的前提。

GLUE 数据集。 我们在 GLUE 基准的八个数据集上评估了 BERT-base [19], BERT-large [19] 和 BART-base [49] ，但由于篇幅限制，仅在图 6 中展示 CoLA、SST-2、MNLI、QQP 和 MRPC 数据集上的结果。对于 BERT-base 模型，在全部八个数据集上，与原始全精度模型相比，我们的 4位后训练量化方法的准确率下降均小于 1%，并且优于所有被研究的方法，包括 4 位、6 位和 8 位训练后量化方法以及量化感知训练方法。由于 GOBO [85] 仅量化权重，因此我们采用相同的方法与其进行比较，结果如表 7 所示。在仅权重量化设置下，我们的方法也优于 GOBO。此外，我们还评估了 BERT-large 模型。由于参数数量更多，与 BERT-base 相比，其挑战性要大得多，因此只有少数先前的量化工作对其进行了评估。表 6 中的结果显示，在所展示的五个数据集上，BERT-large 的准确率损失约为 1%，在其他数据集上也发现了类似结果。对于 BART-base 模型，表 6 中我们的 4位后训练量化结果显示，在所有数据集上，与原始全精度模型的准确率相比，准确率损失约为 2%。在上述评估中，我们的 4位后训练量化结果优于先前工作报告的所有训练后量化结果以及大多数量化感知训练结果。

SQuAD 数据集。 我们还评估了 OliVe 量化在摘要任务 SQuAD [68], 上的准确率。该任务比之前的 GLUE 数据集更具挑战性。表 8 展示了在 SQuAD v1.1 和 SQuAD v2.0 数据集上的结果。在这两个数据集上，我们的

4位后训练量化方法在 BERT-base 模型上的准确率损失低于 2%，在 BART-base 模型上的准确率损失约为 3%，优于最先进的量化工作离群值抑制中的 6位训练后量化方法。

大语言模型。 我们在训练后量化设置下评估 OliVe 对大语言模型的准确率。大语言模型的推理颇具挑战，因为它需要大量内存，这使得重新训练更加消耗资源。因此，对于大语言模型而言，无需重新训练的训练后量化方法比量化感知训练方法更可取。

近期研究 [18] 表明，当 OPT 模型的参数数量增至 6.7B 时，int8量化会导致准确率显著下降。如表 9 所示，我们的 8 位训练后量化方法在 OPT-6.7B 上仅使困惑度略微增加（越低越好），而基于 int8 的量化方法准确率显著下降，在 C4数据集上的表现甚至不如我们的 4位后训练量化方法。对于 GPT2-XL 和 BLOOM-7B1 模型，我们的 8 位训练后量化方法基本达到原始困惑度，而 4位后训练量化方法的性能接近 int8。相比之下，int4 和 4 位 ANT 的准确率结果无法接受（比单精度浮点数模型差 10-1000×）。

总而言之，我们的 OliVe 量化框架将 4位量化 推向了新的最先进水平，能够在大多数数据集上，使 BERT-base、BERT-large 和 BART-base 等常用语言模型的准确率几乎达到原始水平。此外，OliVe 在 GPT2-XL、BLOOM-7B1 和 OPT-6.7B 等大型语言模型上，也实现了最先进的 4位和 8位量化结果。

5.3 图形处理器性能与能耗

我们在 GPU 模拟器上评估大语言模型，其中 GPT 类模型的批量大小设为 2，BERT 类模型的批量大小设为 16。对于 OliVe，4位量化可将损失限制在相对较小的范围内。GOBO [85] 能够达到所有模型的原始准确率，但在动态随机存取存储器中压缩权重时开销显著。请注意，GOBO 仅量化权重张量，并使用 FP16 进行计算。我们在图形处理器中实现了 GOBO 的内存组织。对于 ANT [32], ，我们通过训练后量化设置下的混合精度，使所有模型的准确率或困惑度接近原始水平（BERT 类模型 [19, 49] 的损失为 $< 1\%$，GPT 类模型 [66, 70, 90] 的困惑度为 < 3）。此外，我们还将图形处理器原生的 int8 纳入比较；其准确率损失不可接受，因此仅用于与图形处理器基线进行性能和

方法	Bits	MNLI	STSB(皮尔逊相关系数)
BERT_{base}	32	84.94	89.70
我们的方法（仅权重）	**4**	**84.75**	**89.62**
GOBO*（仅权重）	4	84.45	88.33

表7：在 MNLI 和 STSB 数据集上与 GOBO 的比较。 *我们实现的 GOBO 的准确率与原论文 [85]报告的数量略有不同。

方法	Bits	SQuAD v1.1	SQuAD v2.0
BERT_{base}	32	88.28/80.82	77.34/73.60
Ours	**4**	**86.38/78.16**	**75.90/72.08**
离群值抑制	6	84.48/75.53	74.69/70.55
BART_{base}	32	91.63/84.79	80.82/77.41
Ours	**4**	**88.15/79.87**	**77.37/73.69**
离群值抑制	6	83.68/75.34	74.44/70.36

表8：SQuAD 数据集上的训练后量化结果。

能耗比较。我们将集成 OliVe设计的图形处理器架构与各种基线进行比较。性能和能耗结果如图9所示。

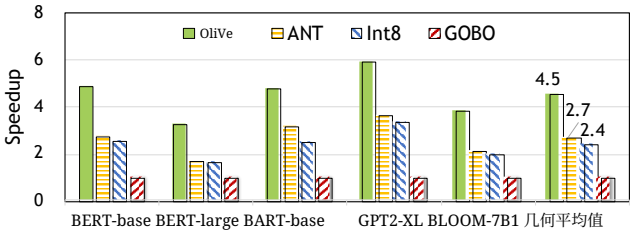
性能。 图9a比较了不同量化方法在 GPU 上的加速比。OliVe 的性能最佳，且在较大的语言模型上比 GOBO 实现了更高的加速比。由于采用 FP16计算和仅权重量化，GOBO [85] 在所有研究的设计中性能最低。相比之下，OliVe 将激活值和权重都量化为低位宽，且不会增加内存访问开销。因此，当参数数量增加时，性能不会下降。由于无法处理离群值，训练后量化严重降低了 ANT [32] 的准确率。在 ANT 中，最终有 80% 的层采用 int8量化，因此 ANT 与 int8 的性能结果相近。平均而言，OliVe 相较于 GOBO、int8 和 ANT 的加速比分别为 4.5×、2.7× 和 2.4× 。

能耗。 图9b展示了不同设计的归一化能耗对比，包括常量功耗、静态功耗和动态功耗。动态功耗包括动态随机存取存储器、二级缓存、一级数据缓存、共享内存、寄存器文件和处理单元（CUDA核心和张量核心）的功耗。一级数据缓存的能耗包含一级缓存和共享内存的能耗总和。得益于已对齐的 4 位设计和对图形处理器的兼容性，OliVe 的能耗最低。由于混合精度的准确率较差，ANT 的能耗也接近 int8。总体而言，4 位 OliVe 非常硬件友好，因此能够充分利用低位宽带来的能耗节省。与 GOBO、int8 和 ANT 相比，OliVe 的平均能耗分别降低了 4.0×、2.3× 和 2.0× 倍。

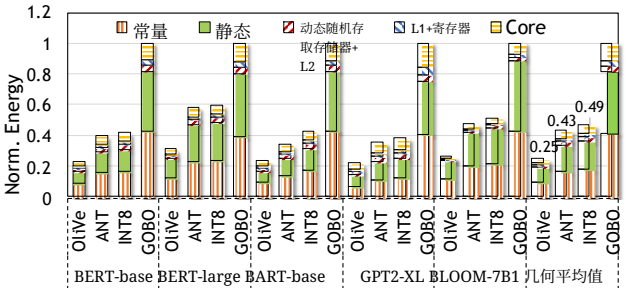
面积。为了测量 OliVe 解码器在图形处理器上的开销，我们将 OliVe 解码器缩放至 12 *nm*，其制造工艺与 RTX 2080 Ti [59] 相同，并计算芯片面积。根据表5，芯片上有 139,264 个 4 位解码器和 69,632 个 8位解码器

方法	GPT2-XL		BLOOM-7B1		OPT-6.7B	
	Wiki	C4	Wiki	C4	Wiki	C4
FP32	17.48	16.30	13.05	14.94	22.14	10.63
int8	18.29	17.35	14.04	16.18	37.45	74.30
8位 OliVe	17.49	16.37	13.13	15.04	22.34	10.73
int4	1E+4	9E+3	3E+6	9E+6	5E+2	1E+2
4 位 ANT	27.79	27.35	23.22	27.36	4E+4	4E+4
4 位 OliVe	19.11	18.08	15.16	17.18	55.44	32.41

表 9：大型语言模型上的训练后量化结果。准确率指标为困惑度，数值越低越好。

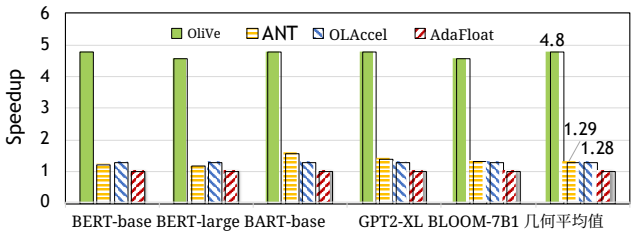


（a）图形处理器上的加速比。

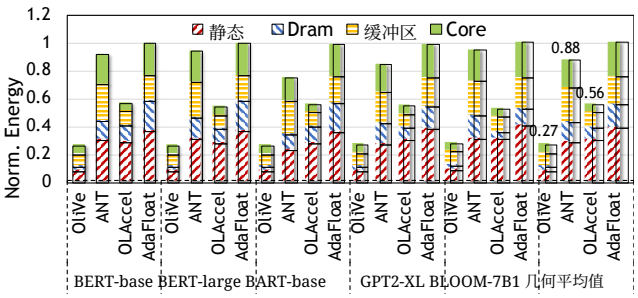


（b）图形处理器上的归一化能耗。

图 9：GPU 上四种不同设计的比较。



(a) 硬件加速器上的加速比。



(b) 硬件加速器上的归一化能耗。

图10：不同设计在加速器上的比较。

，其面积如表10所示。由于 RTX 2080 Ti 的芯片面积为 754 mm^2，4 位解码器和 8 位解码器分别仅占整个图形处理器面积的 0.250% 和 0.166%，我们认为这点开销微不足道且值得付出。

5.4 加速器性能与能耗

如第 5.1 节所述，我们还将 OliVe 集成到基于脉动阵列的硬件加速器中，并将其性能和能耗与现有设计 ANT [32], OLAccel [61], 和 AdaFloat [76]进行比较。与其 GPU 实现类似，ANT 是一种混合精度设计。由于 AdaFloat 不支持混合精度，我们仅提供 8 位量化结果。所有加速器在所有 Transformer模型上都能达到与原始模型接近的准确率。

性能。如图 10a 所示，OliVe 在延迟加速比方面具有最显著的优势。由于 ANT 无法处理离群值，因此在大多数模型上的性能与 OLAccel 相近。OliVe 在所有模型上的加速比数值都非常接近，且不会随着模型参数数量的增加而变化。平均而言，OliVe 相较于 AdaFloat、OLAccel 和 ANT 的加速比分别达到 4.8×、3.8× 和 3.7× 倍。

能耗。图 10b 展示了不同设计的归一化能耗，由静态能耗和动态能耗（动态随机存取存储器、片上缓冲区和核心）组成。OliVe 的能耗最低。与 OLAccel 相比，OliVe 在.....方面具有显著优势

组件	数量	面积（mm^2）	面积占比
4位解码器（13.53μm^2）	139,264	1.88	0.250%
8 位解码器（18.00μm^2）	69,632	1.25	0.166%

表10：OliVe解码器在 RTX 2080 Ti 上的面积。

静态存储和动态随机存取存储器的能耗。混合精度效果较差会增加 ANT 的能耗，在 BLOOM-7B1 模型上甚至接近 AdaFloat。平均而言，OliVe 相较于 AdaFloat、OLAccel 和 ANT 的能耗降低分别达到 3.7×、2.1× 和 3.3× 倍。

面积。表11展示了采用 22 nm 工艺的 OliVe 脉动阵列架构的面积细分。在这种情况下，4 位和 8 位解码器分别带来约占核心面积 2.2% 和 1.5% 的开销；与阵列中处理单元的面积相比，这一开销微不足道。考虑片上存储结构后，总体面积开销还会更小。此外，我们还使用 DeepScaleTool [69] 将其他加速器缩放至 22 nm，并得到了与这些数字相近的结果。请注意，我们实现的所有加速器面积大小相近。OliVe 的面积开销较小，直接得益于精心设计的离群值-牺牲值对（OVP）编码。

6 相关工作与讨论

本节介绍并讨论了有关深度神经网络加速与压缩的研究。随着深度神经网络模型的计算需求不断增长，设计算法和架构以加速深度神经网络模型至关重要。人们提出了各种压缩方法，例如

组件	数量	面积（mm^2）	面积占比
4位解码器 (37.22μm^2)	128	0.00476	2.2%
8 位解码器 (49.50μm^2)	64	0.00317	1.5%
4 位处理单元 (50.01μm^2)	4096	0.205	96.3%

表11：OliVe 在 22 nm process 下的面积细分。

剪枝和量化，以利用深度神经网络的冗余特性。

DNN加速。近年来，人们提出了各种架构 [10, 12, 13, 23, 25, 34, 35, 51, 63, 64, 87, 96]，以匹配深度神经网络模型的计算特征。为了加速深度神经网络系统，大多数优化都聚焦于编译 [11, 40, 92, 95, 97] 和调度 [8, 9, 15 17, 31, 52 54, 84]——。

DNN加速高度依赖矩阵乘法的性能。因此，一些研究通过针对矩阵乘法设计的数据流架构 [10, 12, 25, 34, 35, 41, 63, 64, 87, 96] [41]，着力提升数据复用率并简化控制逻辑。TPU采用了高度优化的数据流架构，能够在多个计算阶段间高效复用数据。现代GPU [60] 现已集成矩阵乘法加速器，例如针对SIMD操作优化的张量核心，从而进一步提升DNN工作负载加速能力。

剪枝。剪枝是指移除深度神经网络层中的部分权重、输入或输出，从而得到模型规模更小的稀疏模型。然而，大幅削减会导致不规则的内存访问，不利于加速推理和训练。为解决这一问题，研究人员在算法和硬件架构方面提出了多种稀疏优化方法，以减少低效计算 [2, 26–29, 36, 64, 65, 80, 89, 94, 98]。此外，NVIDIA Ampere GPU架构中引入了稀疏张量核心 [1]，以支持 2:4 结构化稀疏性。

量化。量化是降低深度神经网络模型规模和计算负担的另一种有效且高效的方法。常见的量化方法有两种，即量化感知训练（QAT）[38, 50, 81, 99] 和训练后量化（PTQ）[30, 35, 38, 81]。QAT通过重新训练，使模型能够适应量化噪声。PTQ的实现非常高效，因为它无需训练数据和训练流程，就能将原始单精度浮点数模型直接转换为低比特模型。因此，PTQ更适用于十亿规模的语言模型。

通过将数据量化为低位宽，量化加速器可以显著降低内存带宽需求并提升计算速度。BitFusion [72] 组合低位宽处理单元，以支持不同位宽的量化。OLAccel [61] 在第一层使用 16 位乘加运算，在其他层使用 4 位乘加运算。DRQ [75] 以不同精度量化敏感区域和不敏感区域，这与离群值感知量化类似。GOBO [85] 是一种利用离群值感知量化的加速器，它以更高精度量化权重中的离群值。然而，上述离群值感知量化加速器存在非对齐内存访问，导致额外开销并限制计算速度。ANT [32]提出了一种定长自适应量化框架，但只考虑张量的分布，忽略了离群值的重要性。相比之下，我们提出的新型 OliVe 量化框架能够以内存对齐且硬件友好的方式处理离群值。

AdaptivFloat [76] 与 abfloat 类似，都是通过给指数添加偏置，但二者的动机以及偏置的确定方式有所不同。AdaptivFloat 旨在适应不同层的动态范围，并使用其算法，以层为粒度计算最优偏置。我们的 abfloat 旨在充分利用编码范围，因此只需给所有编码值添加统一偏置，跳过正常值的范围，实现起来更简单。

GPU架构。英伟达一直在更新其新一代 GPU，例如，Ampere 架构 [1],新增了用于深度神经网络结构化稀疏性的稀疏张量核心，并引入计算数据压缩以提高内存访问带宽。张量核心的结构化稀疏性与我们提出的量化正交，因为我们的逐元素量化不会影响（稀疏）张量核心数据流。Ampere GPU 的计算数据压缩可以压缩动态随机存取存储器和二级缓存中的零值及相似字节。因此，它是无损且通用的。它也对 OliVe 透明且与之正交，因为 OliVe 不会修改存储系统。相比之下，先前的量化工作 [85]在动态随机存取存储器层面执行压缩，因此可能会受到 Ampere GPU 数据压缩的影响。

另一方面，深度神经网络量化是一种有损压缩。我们认为，严格无损的压缩对深度神经网络量化的益处有限。因此，我们的工作可以作为一种专用解决方案，与 Ampere 当前的计算数据压缩互为补充。由于现有的 GPU模拟器 [3, 45]无法支持数据压缩，我们将在未来的工作中继续跟进并研究这一问题。

7 结论

在本研究中，我们提出了一种新颖的离群值-受害值对（OVP）量化方法，能够以较低的硬件开销处理离群值，并实现显著的性能提升。其核心思想是牺牲紧邻关键离群值的正常值（称为受害值），以容纳这些离群值。基于这一思想设计的 OVP 编码能够使离群值与正常值在全局上相同、在局部上可区分。据我们所知，OliVe 将 4 位量化推向了新的最先进水平，能够在常用语言模型上实现近乎原始的准确率。此外，我们的架构设计可以高效集成到现有硬件加速器中，例如张量核心和脉动阵列。最后，基于OliVe的加速器分别以 4.5× 的加速比和 4.0× 的能耗降低幅度，超越了现有的离群值感知加速器 GOBO。

致谢

本研究得到国家重点研发计划（项目编号：2022YFB4501401）以及国家自然科学基金（NSFC）项目（62222210、62072297 和 61832006）的资助。作者感谢匿名审稿人提出的建设性意见，以帮助改进本研究。我们还感谢 Tailong Wangliu 和 Shuangjie Ruan 一直以来的支持。

参考文献

[1] 2020。英伟达 Ampere 架构白皮书。https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf。[2] Jorge Albericio、Patrick Judd、Tayler Hetherington、Tor Aamodt、Natalie Enright Jerger 和 Andreas Moshovos。2016。Cnvlutin：无无效神经元深度神经网络计算。*ACM SIGARCH*《计算机体系结构新闻》44，3（2016），1–13。[3] Ali Bakhoda、George Yuan、Wilson Fung、Henry Wong 和 Tor Aamodt。2009。使用详细的 GPU 模拟器分析 CUDA 工作负载。*ISPASS 2009 -*

国际系统与软件性能分析研讨会，163 – 174 10 1109 2009 4919648 。https://doi.org/ . /ISPASS. . Ron [4] Banner、Yury Nahshan 和 Daniel Soudry。2019。用于快速部署的卷积网络训练后 4 位量化。神经信息处理系统进展32（2019）。[5] Yoshua Bengio、Nicholas Léonard 和 Aaron Courville。2013。针对条件计算，通过随机神经元估计或传播梯度。*arXiv* 预印本 *arXiv:1308.3432*（2013）。[6] Yaohui Cai、Zhewei Yao、Zhen Dong、Amir Gholami、Michael W Mahoney 和 Kurt Keutzer。2020。Zeroq：一种新颖的零样本量化框架。载于*IEEE/CVF*计算机视觉与模式识别会议论文集。13169–13178。[7] Zhaowei Cai 和 Nuno Vasconcelos。2020。重新思考混合精度神经网络的可微搜索。载于*IEEE/CVF*计算机视觉与模式识别会议论文集。2349–2358。[8] Quan Chen、Hailong Yang、Minyi Guo、Ram Srivatsa Kannan、Jason Mars 和 Lingjia Tang。2017。Prophet：面向非抢占式加速器的精准 QoS预测，以提高仓库规模计算机的利用率。载于第二十二届编程语言与操作系统架构支持国际会议论文集，*ASPLOS 2017*，中国西安，*2017*年*4*月*8*日至*12*日。ACM， 17 32 10 1145 3037697 3037700 – 。https://doi.org/ . / . Quan [9] Chen、Hailong Yang、Jason Mars 和 Lingjia Tang。2016。Baymax：为仓库规模计算机中的非抢占式加速器提供 QoS感知并提高利用率。载于第二十一届编程语言与操作系统架构支持国际会议论文集，*ASPLOS 2016*，美国佐治亚州亚特兰大，*2016*年*4*月*2*日至*6*日。ACM，681–696。https://doi.org/10.1145/2872362.2872368[10] Tianshi Chen、Zidong Du、Ninghui Sun、Jia Wang、Chengyong Wu、Yunji Chen 和 Olivier Temam。2014。Diannao：一种小型高吞吐量加速器，适用于普及型机器学习。*ACM SIGARCH*《计算机体系结构新闻》42，1（2014），269–284。[11] Tianqi Chen、Thierry Moreau、Ziheng Jiang、Lianmin Zheng、Eddie Q. Yan、Haichen Shen、Meghan Cowan、Leyuan Wang、Yuwei Hu、Luis Ceze、Carlos Guestrin 和 Arvind Krishnamurthy。2018。TVM：面向深度学习的自动化端到端优化编译器。载于第*13*届 *USENIX* 操作系统设计与实现研讨会，*OSDI 2018*，*2018*年*10*月 578 594 10 5555 3291168 3291211美国加利福尼亚州卡尔斯巴德，*10*月 -。USENIX 协会，– 。https://doi.org/ . / .[12] Yunji Chen、Tao Luo、Shaoli Liu、Shijin Zhang、Liqiang He、Jia Wang、Ling Li、Tianshi Chen、Zhiwei Xu、Ninghui Sun 等。2014。Dadiannao：一台机器学习超级计算机。载于*2014*年第*47*届*IEEE/ACM*微体系结构国际研讨会。IEEE，609–622。[13] Yu-Hsin Chen、Tushar Krishna、Joel S Emer 和 Vivienne Sze。2016。Eyeriss：一种适用于深度卷积神经网络的节能型可重构加速器。*IEEE* 固态电路期刊52，1（2016），127–138。[14] Jungwook Choi、Zhuo Wang、Swagath Venkataramani、Pierce I-Jen Chuang、Vijayalakshmi Srinivasan 和 Kailash Gopalakrishnan。2018。Pact：面向量化神经网络的参数化裁剪激活。*arXiv* 预印本 *arXiv:1805.06085*（2018）。[15] Yujeong Choi、Yunseong Kim 和 Minsoo Rhu。2021。惰性批处理：面向云端机器学习推理的 SLA感知批处理系统。载于*IEEE*高性能计算机体系结构国际研讨会，*HPCA 2021*，韩国首尔，*2021*年*2*月*27*日至*3*月*3*日。IEEE，493–506。https: //doi.org/10.1109/HPCA51647.2021.00049 [16] Weihao Cui、Mengze Wei、Quan Chen、Xiaoxin Tang、Jingwen Leng、Li Li 和 Mingyi Guo。2019。Ebird：弹性批处理，提升深度学习服务的响应能力和吞吐量。载于第*37*届*IEEE*国际计算机设计会议，*ICCD 2019*，*2019*年*10*月497 505 10 1109 46524 2019 00075阿联酋阿布扎比，*11*月 -。IEEE，– 。https://doi.org/ . /ICCD . .[17] Weihao Cui、Han Zhao、Quan Chen、Hao Wei、Zirui Li、Deze Zeng、Chao Li 和 Minyi Guo。2022。DVABatch：多样性感知多入口多出口批处理，实现 GPU 上深度神经网络服务的高效处理。载于*2022*年 *USENIX*年度技术会议（*USENIX ATC 22*）。183–198。[18] Tim Dettmers、Mike Lewis、Younes Belkada 和 Luke Zettle moyer。2022。Llm. int8 ()：面向大规模 Transformer 的 8位矩阵乘法。*arXiv* 预印本 *arXiv:2208.07339*（2022）。[19] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。2018。Bert：用于语言理解的深度双向Transformer预训练。*arXiv* 预印本 *arXiv:1810.04805*（2018）。[20] Jesse Dodge、Maarten Sap、Ana Marasović、William Agnew、Gabriel Ilharco、Dirk Groeneveld、Margaret Mitchell 和 Matt Gardner。2021。大型网络文本语料库文档记录：海量清洁爬取语料库案例研究。arXiv:arXiv:2104.08758[21] Zhen Dong、Zhewei Yao、Daiyaan Arfeen、Amir Gholami、Michael W Mahoney 和 Kurt Keutzer。2020。Hawq-v2：神经网络的海森矩阵感知迹加权量化。神经信息处理系统进展33（2020），18518–18529。[22] Zhen Dong、Zhewei Yao、Amir Gholami、Michael W Mahoney 和 Kurt Keutzer。2019。Hawq：采用混合精度的海森感知神经网络量化。载于 IEEE/CVF国际计算机视觉会议论文集。293–302。[23] Zidong Du、Robert Fasthuber、Tianshi Chen、Paolo Ienne、Ling Li、Tao Luo、Xiaobing Feng、Yunji Chen 和 Olivier Temam。2015。ShiDianNao：将视觉处理移近传感器。载于第42届年度国际计算机体系结构研讨会论文集。92–104。[24] Amir Gholami、Zhewei Yao、Sehoon Kim、Michael W Mahoney 和 Kurt Keutzer。2021。人工智能与内存墙。RiseLab Medium文章（2021）。[25] Vinayak Gokhale、Jonghoon Jin、Aysegul Dundar、Berin Martini 和 Eugenio Culurciello。2014。用于深度神经网络的240 G-ops/s移动协处理器。载于 IEEE计算机视觉与模式识别会议研讨会论文集。682–687。[26] Yue Guan、Jingwen Leng、Chao Li、Quan Chen 和 Minyi Guo。2020。BERT关注的范围有多远：BERT注意力的基于距离的聚类与分析。arXiv预印本 arXiv:2011.00943（2020）。[27] Yue Guan、Zhengyi Li、Jingwen Leng、Zhouhan Lin 和 Minyi Guo。2022。Tran-skimmer：Transformer学习逐层跳读。arXiv预印本 arXiv:2205.07324（2022）。[28] Yue Guan、Zhengyi Li、Zhouhan Lin、Yuhao Zhu、Jingwen Leng 和 Minyi Guo。2022。Block-skim：面向Transformer的高效问答。载于 AAAI人工智能会议论文集，第36卷。10710–10719。[29] Cong Guo、Bo Yang Hsueh、Jingwen Leng、Yuxian Qiu、Yue Guan、Zehuan Wang、Xiaoying Jia、Xipeng Li、Minyi Guo 和 Yuhao Zhu。2020。借助分块稀疏性，在无硬件支持的情况下加速稀疏DNN模型。载于 SC20：高性能计算、网络、存储与分析国际会议。IEEE，1–15。[30] Cong Guo、Yuxian Qiu、Jingwen Leng、Xiaotian Gao、Chen Zhang、Yunxin Liu、杨帆、Yuhao Zhu 和 Minyi Guo。2022。SQuant：通过对角海森近似实现即时无数据量化。载于国际学习表征会议。https://openreview.net/forum?id=JXhROKNZzOc[31] Cong Guo、Yuxian Qiu、Jingwen Leng、Chen Zhang、Ying Cao、Quanlu Zhang、Yunxin Liu、杨帆 和 Minyi Guo。2022。嵌套前向自动微分，实现内存高效的深度神经网络训练。载于 2022 IEEE第40届国际计算机设计会议（ICCD）。IEEE，738–745。[32] Cong Guo、Chen Zhang、Jingwen Leng、Zihan Liu、杨帆、Yunxin Liu、Minyi Guo 和 Yuhao Zhu。2022。ANT：利用自适应数值数据类型实现低比特深度神经网络量化。载于2022年第55届IEEE/ACM国际微体系结构研讨会（MICRO）。IEEE，1414–1433。[33] Cong Guo、Chen Zhang、Jingwen Leng、Zihan Liu、杨帆、Yunxin Liu、Minyi Guo 和 Yuhao Zhu。2022。ANT GitHub代码库。https://github.com/clevercool/ ANT Micro22。_[34] Cong Guo、Yangjie Zhou、Jingwen Leng、Yuhao Zhu、Zidong Du、Quan Chen、Chao Li、Bin Yao 和 Minyi Guo。2020。通过时分GPU-脉动阵列集成平衡DNN加速的效率与灵活性。载于 2020年第57届ACM/IEEE设计自动化会议（DAC）。1–6。[35] Suyog Gupta、Ankur Agrawal、Kailash Gopalakrishnan 和 Pritish Narayanan。2015。有限数值精度下的深度学习。载于国际机器学习会议。PMLR，1737–1746。[36] Song Han、Huizi Mao 和 William J Dally。2015。深度压缩：通过剪枝、训练量化和霍夫曼编码压缩深度神经网络。arXiv预印本 arXiv:1510.00149（2015）。[37] Kaiming He、Xiangyu Zhang、Shaoqing Ren 和 Jian Sun。2016。用于图像识别的深度残差学习。载于 IEEE计算机视觉与模式识别会议论文集。770–778。[38] Benoit Jacob、Skirmantas Kligys、Bo Chen、Menglong Zhu、Matthew Tang、Andrew Howard、Hartwig Adam 和 Dmitry Kalenichenko。2018。面向高效仅整数运算推理的神经网络量化与训练。载于 IEEE计算机视觉与模式识别会议论文集。2704–2713。[39] Shubham Jain、Swagath Venkataramani、Vijayalakshmi Srinivasan、Jungwook Choi、Kailash Gopalakrishnan 和 Leland Chang。2019。BiScaled-DNN：利用两个缩放因子量化深度神经网络中的长尾数据结构。载于2019年第56届ACM/IEEE设计自动化会议（DAC）。IEEE，1–6。[40] Zhihao Jia、Oded Padon、James J. Thomas、Todd Warszawski、Matei Zaharia 和 Alex Aiken。2019。TASO：通过自动生成图替换优化深度学习计算。载于第27届ACM操作系统原理研讨会论文集（SOSP）。ACM，47–62。https://doi.org/10.1145/ 3341301.3359630[41] Norman P. Jouppi、Doe Hyun Yoon、Matthew Ashcraft、Mark Gottscho、Thomas B. Jablin、George Kurian、James Laudon、Sheng Li、Peter Ma、Xiaoyu Ma、Thomas Norrie、Nishant Patil、Sushma Prasad、Cliff Young、Zongwei Zhou 和 David Patterson。2021。塑造 Google TPUv4i的三代产品带来的十条经验：工业产品。载于2021年ACM/IEEE第48届年度国际计算机体系结构研讨会（ISCA）。[42] Norman P Jouppi、Cliff Young、Nishant Patil、David Patterson、Gaurav Agrawal、Raminder Bajwa、Sarah Bates、Suresh Bhatia、Nan Boden、Al Borchers 等。2017。

数据中心内张量处理单元的性能分析。载于第44届年度国际计算机体系结构研讨会论文集。1–12。[43] Sangil Jung、Changyong Son、Seohyung Lee、Jinwoo Son、Jae-Joon Han、Youngjun Kwak、Sung Ju Hwang 和 Changkyu Choi。2019。通过优化任务损失中的量化区间学习量化深度网络。载于 IEEE/CVF计算机视觉与模式识别会议论文集。4350–4359。[44] Andrew Kerr、Haicheng Wu、Manish Gupta、Dustyn Blasig、Pradeep Ramini、Duane Merrill、Aniket Shivam、Piotr Majcher、Paul Springer、Markus Hohnerbach、Jin Wang 和 Matt Nicely。2022。CUTLASS。https://github.com/NVIDIA/cutlass [45] Mahmoud Khairy、Zhesheng Shen、Tor Aamodt 和 Timothy Rogers。2020。Accel-Sim：用于经验证GPU建模的可扩展仿真框架。473–486。https://doi.org/10 1109 45697 2020 00047 。/ISCA。。Mahmoud Khairy、[46] Zhesheng Shen、Tor M Aamodt 和 Timothy G Rogers。2020。Accel-Sim：用于经验证GPU建模的可扩展仿真框架。载于2020年ACM/IEEE第47届年度国际计算机体系结构研讨会（ISCA）。IEEE，473–486。[47] Pran Kurup 和 Taher Abbasi。2012。使用Synopsys®进行逻辑综合。Springer Science & Business Media。[48] Jingwen Leng、Tayler Hetherington、Ahmed ElTantawy、Syed Gilani、Nam Kim、Tor Aamodt 和 Vijay Janapa Reddi。2013。GPUWattch：实现通用GPU中的能耗优化。ACM SIGARCH《计算机体系结构新闻》41（07 2013）。https://doi.org/10.1145/2508148.2485964[49] Mike Lewis、Yinhan Liu、Naman Goyal、Marjan Ghazvininejad、Abdelrahman Mohamed、Omer Levy、Ves Stoyanov 和 Luke Zettlemoyer。2019。BART：面向自然语言生成、翻译和理解的去噪序列到序列预训练。arXiv预印本 arXiv:1910.13461（2019）。[50] Zhengyi Li、Cong Guo、Zhanda Zhu、Yangjie Zhou、Yuxian Qiu、Xiaotian Gao、Jingwen Leng 和 Minyi Guo。2022。通过自适应舍入边界实现高效激活量化，用于训练后量化。arXiv预印本 arXiv:2208.11945（2022）。[51] Daofu Liu、Tianshi Chen、Shaoli Liu、Jinhong Zhou、Shengyuan Zhou、Olivier Teman、Xiaobing Feng、Xuehai Zhou 和 Yunji Chen。2015。PuDianNao：一种多用途机器学习加速器。载于第二十届编程语言与操作系统架构支持国际会议论文集。369–381。[52] Zihan Liu、Jingwen Leng、Zhihui Zhang、Quan Chen、Chao Li 和 Minyi Guo。2022。VELTAIR：通过自适应编译与调度实现高性能多租户深度学习服务。载于 ASPLOS '22：第27届ACM编程语言与操作系统架构支持国际会议，瑞士洛桑，2022年2月28日—3月4日，Babak Falsafi、Michael Ferdman、Shan Lu 和 Thomas F. Wenisch（编）。ACM，388–401。https: //doi.org/10.1145/3503222.3507752[53] David Lo、Liqun Cheng、Rama Govindaraju、Parthasarathy Ranganathan 和 Christos Kozyrakis。2015。Heracles：大规模提升资源效率。载于第42届年度国际计算机体系结构研讨会（ISCA）论文集。https://doi.org/10 1145 2749469 2749475 。/。[54] Jason Mars、Lingjia Tang、Robert Hundt、Kevin Skadron 和 Mary Lou Soffa。2011。Bubble-Up：通过合理的协同部署提升现代仓储规模计算机的利用率。载于IEEE/ACM微体系结构国际研讨会（MICRO）。https://doi.org/10 1145 2155620 2155650 。/。ModelTC。。代码库。[55] https://huggingface.co/ModelTC。[56] Naveen Muralimanohar、Rajeev Balasubramonian 和 Norman P Jouppi。2009。CACTI 6.0：用于建模大型缓存的工具。惠普实验室27（2009），28。[57] Markus Nagel、Marios Fournarakis、Rana Ali Amjad、Yelysei Bondarenko、Mart van Baalen 和 Tijmen Blankevoort。2021。神经网络量化白皮书。arXiv预印本 arXiv:2106.08295（2021）。[58] 英伟达。2017。NVIDIA Tesla V100 GPU架构。载于技术报告。NVIDIA。[59] 英伟达。2018。NVIDIA Turing GPU架构。载于技术报告。NVIDIA。[60] 英伟达。2020。NVIDIA A100张量核心架构。载于技术报告。NVIDIA。[61] Eunhyeok Park、Dongyoung Kim 和 Sungjoo Yoo。2018。基于异常值感知低精度计算的节能神经网络加速器。载于2018年ACM/IEEE第45届年度国际计算机体系结构研讨会（ISCA）。IEEE，688–698。[62] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。2019。PyTorch：一种命令式风格、高性能深度学习库。神经信息处理系统进展32（2019），8026–8037。[63] Maurice Peemen、Arnaud AA Setio、Bart Mesman 和 Henk Corporaal。2013。面向卷积神经网络的以内存为中心的加速器设计。载于2013年IEEE第31届计算机设计国际会议（ICCD）。IEEE，13–19。[64] Eric Qin、Ananda Samajdar、Hyoukjun Kwon、Vineet Nadella、Sudarshan Srinivasan、Dipankar Das、Bharat Kaul 和 Tushar Krishna。2020。Sigma：一种用于DNN训练、带有灵活互连的稀疏且不规则的通用矩阵乘法加速器。载于2020年IEEE高性能计算机体系结构国际研讨会（HPCA）。IEEE，58–70。[65] Yuxian Qiu、Jingwen Leng、Cong Guo、Quan Chen、Chao Li、Minyi Guo 和 Yuhao Zhu。2019。通过网络分析驱动的路径提取实现对抗防御。载于 IEEE/CVF计算机视觉与模式识别会议论文集以及模式识别（CVPR）。[66] Alec Radford、Jeff Wu、Rewon Child、David Luan、Dario Amodei 和 Ilya Sutskever。2019。《语言模型是无监督的多任务学习者》。（2019）。[67] Md Aamir Raihan、Negar Goli 和 Tor M Aamodt。2019。深度学习加速器赋能 GPU 的建模。载于《2019 IEEE系统与软件性能分析国际研讨会》（ISPASS）。IEEE，79–92。[68] Pranav Rajpurkar、Jian Zhang、Konstantin Lopyrev 和 Percy Liang。2016。SQuAD：100，000+ 用于文本机器阅读理解的问题。载于《2016年自然语言处理实证方法会议论文集》。计算语言学协会。[69] Satyabrata Sarangi 和 Bevan Baas。2021。DeepScaleTool：用于深亚微米时代技术缩放精确估算的工具。载于《2021 IEEE电路与系统国际研讨会》（ISCAS）。IEEE，1–5。[70] Teven Le Scao、Angela Fan、Christopher Akiki、Ellie Pavlick、Suzana Ilić、Daniel Hesslow、Roman Castagné、Alexandra Sasha Luccioni、François Yvon、Matthias Gallé 等。2022。BLOOM：拥有1760亿参数的开放获取多语言语言模型。arXiv预印本 arXiv:2211.05100（2022）。[71] Hardik Sharma、Jongse Park、Divya Mahajan、Emmanuel Amaro、Joon Kyung Kim、Chenkai Shao、Asit Mishra 和 Hadi Esmaeilzadeh。2016。从高级深度神经网络模型到现场可编程门阵列。载于《2016年第49届IEEE/ACM微体系结构国际研讨会》（MICRO）。IEEE，1–12。[72] Hardik Sharma、Jongse Park、Naveen Suda、Liangzhen Lai、Benson Chau、Vikas Chandra 和 Hadi Esmaeilzadeh。2018。位融合：用于加速深度神经网络的位级动态可组合架构。载于《2018年ACM/IEEE第45届年度国际计算机体系结构研讨会》（ISCA）。IEEE，764–775。[73] Hardik Sharma、Jongse Park、Naveen Suda、Liangzhen Lai、Benson Chau、Vikas Chandra 和 Hadi Esmaeilzadeh。2018。Bitfusion GitHub代码库。https: //github.com/hsharma35/bitfusion。[74] Sheng Shen、Zhen Dong、Jiayu Ye、Linjian Ma、Zhewei Yao、Amir Gholami、Michael W Mahoney 和 Kurt Keutzer。2020。Q-bert：基于海森矩阵的 BERT 超低精度量化。载于《AAAI人工智能会议论文集》第34卷。8815–8821。[75] Zhuoran Song、Bangqi Fu、Feiyang Wu、Zhaoming Jiang、Li Jiang、Naifeng Jing 和 Xiaoyao Liang。2020。DRQ：面向深度神经网络加速的动态区域量化。载于《2020年ACM/IEEE第47届年度国际计算机体系结构研讨会》（ISCA）。IEEE，1010–1021。[76] Thierry Tambe、En-Yu Yang、Zishen Wan、Yuntian Deng、Vijay Janapa Reddi、Alexander Rush、David Brooks 和 Gu-Yeon Wei。2020。面向弹性深度学习推理的自适应浮点编码算法与硬件协同设计。载于《2020年第57届ACM/IEEE设计自动化会议》（DAC）。IEEE，1–6。[77] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。2017。注意力机制就是你所需要的一切。《神经信息处理系统进展》30（2017）。[78] Alex Wang、Amanpreet Singh、Julian Michael、Felix Hill、Omer Levy 和 Samuel R Bowman。2018。GLUE：自然语言理解多任务基准与分析平台。arXiv预印本 arXiv:1804.07461（2018）。[79] Kuan Wang、Zhijian Liu、Yujun Lin、Ji Lin 和 Song Han。2019。Haq：具有混合精度的硬件感知自动量化。载于《IEEE/CVF计算机视觉与模式识别会议论文集》。8612–8620。[80] Yang Wang、Chen Zhang、Zhiqiang Xie、Cong Guo、刘云欣和Jingwen Leng。2021。双侧稀疏张量核心。载于《2021年ACM/IEEE第48届年度国际计算机体系结构研讨会》（ISCA）。IEEE，1083–1095。[81] Ziwei Wang、Jiwen Lu、Chenxin Tao、Jie Zhou 和 Qi Tian。2019。学习二值卷积神经网络中的通道级交互。载于《IEEE/CVF计算机视觉与模式识别会议论文集》。568–577。[82] Xiuying Wei、Yunchen Zhang、Xiangguo Zhang、Ruihao Gong、Shanghang Zhang、Qi Zhang、Fengwei Yu 和 Xianglong Liu。2022。离群值抑制：突破低比特 Transformer 语言模型的极限。载于《神经信息处理系统进展》，Alice H. Oh、Alekh Agarwal、Danielle Belgrave 和 Kyunghyun Cho（编）。https://openreview.net/forum?id=yW5zeRSFdZ [83] 维基百科贡献者。2022。68–95–99.7法则——维基百科，自由的百科全书。 [在线]。[84] Hailong Yang、Alex D. Breslow、Jason Mars 和 Lingjia Tang。2013。Bubble-flux：面向仓储规模计算机、用于提升利用率的精确在线服务质量管理。载于《第40届年度国际计算机体系结构研讨会》（ISCA）。https://doi.org/10 1145 2485922 2485974 。/。[85] Ali Hadi Zadeh、Isak Edo、Omar Mohamed Awad 和 Andreas Moshovos。2020。GOBO：对基于注意力机制的自然语言处理模型进行量化，以实现低延迟、低能耗推理。载于《2020年第53届IEEE/ACM微体系结构国际研讨会》（MICRO）。IEEE，811–824。[86] Ofir Zafrir、Guy Boudoukh、Peter Izsak 和 Moshe Wasserblat。2019。Q8BERT：量化8位BERT。载于《2019年第五届能源高效机器学习与认知计算-NeurIPS专刊研讨会》（EMC2-NIPS）。IEEE，36–39。[87] Chen Zhang、Peng Li、Guangyu Sun、Yijin Guan、Bingjun Xiao 和 Jason Cong。2015。优化面向深度卷积神经网络的基于FPGA的加速器设计。载于《2015年ACM/SIGDA现场可编程门阵列国际研讨会论文集》。161–170。

[88] Dongqing Zhang、Jiaolong Yang、Dongqiangzi Ye 和 Gang Hua。2018。Lq-nets：面向高精度、紧凑型深度神经网络的学习式量化。载于《欧洲计算机视觉会议论文集》（*ECCV*）。365–382。[89] Shijin Zhang、Zidong Du、Lei Zhang、Huiying Lan、Shaoli Liu、Ling Li、Qi Guo、Tianshi Chen 和 Yunji Chen。2016。Cambricon-X：面向稀疏神经网络的加速器。载于《*2016*年第*49*届*IEEE/ACM*微体系结构国际研讨会》（*MICRO*）。IEEE，1–12。[90] Susan Zhang、Stephen Roller、Naman Goyal、Mikel Artetxe、Moya Chen、Shuohui Chen、Christopher Dewan、Mona Diab、Xian Li、Xi Victoria Lin 等。2022。OPT：开放式预训练 Transformer 语言模型。*arXiv*预印本 *arXiv:2205.01068*（2022）。[91] Lianmin Zheng、Chengfan Jia、Minmin Sun、Zhao Wu、Cody Hao Yu、Ameer Haj-Ali、Yida Wang、Jun Yang、Danyang Zhuo、Koushik Sen、Joseph E. Gonzalez 和 Ion Stoica。2020。Ansor：为深度学习生成高性能张量程序。载于《第*14*届*USENIX*操作系统设计与实现研讨会》（*OSDI*）。https://doi.org/10 5555 3488766 3488815 。/ 。Size Zheng、Yun [92] Liang、Shuo Wang、Renze Chen 和 Kaiwen Sheng。2020。Flex-Tensor：用于异构系统张量计算的自动调度探索与优化框架。载于《编程语言与操作系统架构支持会议》，洛桑（*ASPLOS*）。https: //doi.org/10 1145 3373376 3378508 。/ 。Shuchang Zhou、Yuxin Wu、Zekun Ni、Xinyu Zhou、He Wen 和 Yuheng Zou。2016。Dorefa-net：使用低位宽梯度训练低位宽卷积神经网络。*arXiv*预印本 *arXiv:1606.06160*（2016）。[94] [93] Xuda Zhou、Zidong Du、Qi Guo、Shaoli Liu、Chengsi Liu、Chao Wang、Xuehai Zhou、Ling Li、Tianshi Chen 和 Yunji Chen。2018。Cambricon-S：解决通过软硬件协同方法处理稀疏神经网络中的不规则性。载于*2018*年第*51*届 *IEEE/ACM* 微架构年度国际研讨会（*MICRO*）。IEEE，15–28。[95] Yangjie Zhou、Jingwen Leng、Yaoxu Song、Shuwen Lu、Mian Wang、Chao Li、Minyi Guo、Wenting Shen、Yong Li 等。2023。uGrapher：通过统一抽象实现图神经网络的高性能图算子计算。载于第*28*届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第*2*卷。878–891。[96] Yangjie Zhou、Mengtian Yang、Cong Guo、Jingwen Leng、Yun Liang、Quan Chen、Minyi Guo 和朱宇豪。2021。商业矩阵乘法加速器上隐式卷积算法的特征分析与揭秘。载于*2021 IEEE* 工作负载特征分析国际研讨会（*IISWC*）。IEEE，214–225。[97] Hongyu Zhu、Ruofan Wu、Yijia Diao、Shanbin Ke、Haoyu Li、Chen Zhang、Jilong Xue、Lingxiao Ma、Yuqing Xia、Wei Cui、杨帆、Mao Yang、Lidong Zhou、Asaf Cidon 和 Gennady Pekhimenko。2022。ROLLER：面向深度学习的快速高效张量编译。载于第*16*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 22*）。233–248。[98] Maohua Zhu、Tao Zhang、Zhenyu Gu 和 Yuan Xie。2019。稀疏张量核心：面向现代 GPU 上向量级稀疏神经网络的算法与硬件协同设计。载于第*52*届 *IEEE/ACM* 微架构年度国际研讨会论文集。359–371。[99] Bohan Zhuang、Mingkui Tan、Jing Liu、Lingqiao Liu、Ian Reid 和 Chunhua Shen。2021。使用低位宽权重与激活值有效训练卷积神经网络。*IEEE* 模式分析与机器智能汇刊 (2021)。