BitGNN：释放二值图神经网络在 GPU 上的性能潜力

Jou-An Chen 北卡罗来纳州立大学 美国北卡罗来纳州罗利 jchen73@ncsu.edu

Hsin-Hsuan Sung 北卡罗来纳州立大学 美国北卡罗来纳州罗利 hsung2@ncsu.edu

Xipeng Shen 北卡罗来纳州立大学 美国北卡罗来纳州罗利 xshen5@ncsu.edu

Sutanay Choudhury 太平洋西北国家实验室 美国华盛顿州里奇兰 Sutanay.Choudhury@pnnl.gov

Ang Li 太平洋西北国家实验室 美国华盛顿州里奇兰 ang.li@pnnl.gov

arXiv:2305.02522v2 [cs.DC] 3 Jun 2023

摘要

近期研究表明，二值图神经网络（GNN）能够通过二值化张量减少图神经网络的计算量，具有广阔前景。然而，既有工作主要聚焦于算法设计或训练技术，尚未充分探讨如何在加速器硬件上充分发挥其性能潜力。本研究从效率角度重新设计了二值图神经网络推理后端，通过提出一系列抽象和技术，填补了这一空白，使二值图神经网络及其计算能够更好地适应 GPU 上的比特操作特性。在真实世界图数据集上使用图卷积网络（GCN）、GraphSAGE 和 GraphSAINT 进行的实验表明，在保持相同准确率的前提下，所提出的技术比最先进的二值图神经网络实现快 8 至 22 倍。BitGNN 代码已公开提供。[1]。

CCS 概念

• **计算方法** → **神经网络**； **大规模并行算法**； • **计算机系统组织** → **单指令、多数据**。

关键词

图神经网络、二值化 GNN、位操作、GPU、稀疏矩阵

ACM 参考格式：

Jou-AnChen、Hsin-Hsuan Sung、Xipeng Shen、Sutanay Choudhury 和 Ang Li。2023。BitGNN：释放二值图神经网络在 GPU 上的性能潜力。载于国际超算大会（*ICS '23*），*2023*年*6*月*21*日至*23*日，美国佛罗里达州奥兰多。ACM，美国纽约州纽约，13页。https://doi.org/10.1145/3577193.3593725

[1]https://github.com/PICTureRG/BitGNN

允许为个人或课堂使用而免费制作本作品的全部或部分数字副本或纸质副本，但不得为营利或商业利益而制作或分发副本，并且副本首页必须包含此声明和完整引文。对于本作品中由作者以外人员拥有版权的组件，必须尊重其版权。注明出处后可以摘录。以其他方式复制、再版、发布到服务器或分发到列表，均须事先获得具体许可和/或支付费用。请向 permissions@acm.org 申请许可。*ICS '23，2023*年*6*月*21*日至*23*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 ACM。ACM ISBN 979-8-4007-0056-9/23/06...$15.00
https://doi.org/10.1145/3577193.3593725

1 引言

近年来，图神经网络（GNN）在各个领域的应用迅速增长，从社交网络 [1] 到生物信息学 [2], 、计算化学 [3, 4],、3D 计算机视觉 [5, 6], 等。图神经网络使神经网络的线性和非线性变换能够直接作用于图。通过将图结构以及节点/边特征无缝整合到建模过程中，图神经网络天然适用于基于图的问题。大量关于基于图的分类和预测的研究 [7–12] 报告称，图神经网络取得的结果显著优于传统方法。

由于现实世界中的图通常规模庞大，图神经网络推理往往耗时较长且占用大量存储空间，经常超出存储设备或内存的容量以及理想的延迟要求。受普通深度神经网络 [13–17], 二值化（即量化至 1 比特极限）技术的启发，一项新兴研究工作开始探索图神经网络二值化的潜力。通过将激活图、权重和/或邻接矩阵中的每个数值转换为单个比特（有时借助矩阵因子分解实现），图神经网络二值化能够显著降低存储需求和计算量。

近几年来，关于图神经网络二值化的持续研究取得了一些显著进展 [18–21]。例如，二值化导致的精度损失已从 16% 降至不足 5% [18]。这些研究提出了一些方案，能够在各种图和架构上将准确率损失控制在较小范围内，同时在理论上大幅减少计算量和内存用量。然而，如何在真实机器上将这一潜力转化为充分的加速比和内存节省，仍是一个悬而未决的问题。例如，根据理论分析，一项近期研究 [18] 的作者报告称，二值图神经网络可节省 ~47X 的计算量，并可能将内存占用减少 ~30X。但他们的实验表明，与原始的非二值化图神经网络相比，并未实现加速或节省内存。

现有关于二值图神经网络的研究主要聚焦于网络架构 [18–20] 或训练技术 [18, 20, 21]的算法层面 改进。二值图神经网络推理在硬件上的巨大潜力尚未得到充分挖掘。

将张量量化为比特，并在基于字的架构上高效地表示和操作这些张量，其中涉及诸多复杂之处。对于图神经网络而言，情况尤为如此，原因有几个。首先，图神经网络层通常由交替排列的稠密算子和稀疏算子构成。在尽可能降低推理延迟并实现理想的精度与速度权衡的同时，协调稠密与稀疏位级布局之间的不一致并非易事。其次，在二值图神经网络中，激活值/权重中的比特含义（1/0 表示 +/-）与图中的比特含义（1/0 表示连通性）不同。鉴于这些特殊性质，必须谨慎调度整数内在函数，以最大限度地提高位级并行性和线程级并行性。第三，为了充分利用比特表示带来的显著内存节省，必须将寄存器和内存层次结构的精细运用与加速器上的大规模并行性协同起来。

本工作首次对二值图神经网络的性能潜力进行了系统探索。其关注的目标硬件是图形处理器（GPU），这是图神经网络和二值图神经网络最常用的设备。本工作做出了四方面的贡献：

(1) 通过一系列经过比特级优化的抽象来实现二值图神经网络。为有效实现二值图神经网络模型，可以将比特级 BLAS 内核以多种方式组合，并灵活选择精确率、消除重新二值化以及融合方式。我们提出了首个已知的两级二值图神经网络抽象，它对用户隐藏了复杂性，灵活支持各种场景，并可轻松即插即用地将图神经网络转换为二值图神经网络。

（2）位张量的表示。二值 GNN 包含二值化激活值、权重和邻接矩阵。与激活值和权重不同，邻接矩阵通常是稀疏的，其表示面临粒度困境。我们提出一种精细表示动态粗化（FRDC）方案，以同时降低图的存储开销并最大限度地提高 GPU 上的内核效率。此外，我们还提出了几种解决方案，以协调二值图与激活值之间 1/0 和 +/- 的不一致性。

(3) BitGNN 抽象的具体实现。我们通过二值内在函数进行细致的位操作，并对内存和并行性进行优化，将 BitGNN 抽象构建为高效的库。针对不同的精确率需求，我们设计并实现了采用位操作内置函数的抽象。我们提出了一系列技术，专门处理工作负载分解、利用寄存器和片上存储器进行位张量加载/存储、线程束内与线程束间同步，以及高效的二值稀疏矩阵乘法（BSpMM）内核。

(4) BitGNN 的评估。我们将所有技术整合为一个完整的解决方案，即 BitGNN，并以库和调优实用工具集合的形式提供。与最先进的二值图神经网络 [18] 和全精度图神经网络 [22] 实现相比，BitGNN 在 GCN、Graph-SAGE 和 GraphSAINT 模型上分别将延迟最多降低了 26 倍、22 倍和 19 倍。同时，BitGNN 使图神经网络能够高效处理若干此前的解决方案无法处理的超大图。

2 背景章节

2.1 图神经网络

图神经网络中通常有两类操作：聚合和应用。聚合阶段从输入图的几何结构中收集信息。应用阶段则执行典型神经网络中的线性或非线性操作。图神经网络可以形式化表示如下：

$$\mathbf{X}_i^{(l+1)} = \gamma_\theta \left(\mathbf{X}_i^{(l)}, \Psi_{j \in \mathcal{N}(i)} \, \phi_\theta \left(\mathbf{X}_i^{(l)}, \mathbf{X}_j^{(l)}, \mathbf{E}_{i,j} \right) \right)$$

对于节点分类任务，节点表示是每个图神经网络层中的激活值。节点 i 在第 l 层的表示记为 $\mathbf{X}(l)i$。$\mathbf{N}(i)$ 表示节点 i 的邻居集合。在 聚合 阶段，Ψ 表示执行消息聚合的置换不变函数。常用函数包括求和、求平均值和取最大值。在聚合之前，ϕ_θ 表示应用于以下对象的任何用户定义算子：$\mathbf{X}_i$（节点 i 的表示）、$\mathbf{X}j$（节点 j 的表示，即节点 i 的邻居），或 $\mathbf{E}i,j$（边 (i, j) 的表示）。聚合之后，γ_θ 表示 应用 阶段的操作。大多数图神经网络在此阶段包含神经网络操作（例如，多层感知机和 ReLU ）。下面简要介绍几种常见的图神经网络。

GCNConv 图卷积网络（GCN） [7]引入了一种能够直接学习图结构的半监督学习算法。GCNConv 的一个前向层可定义如下：

$$\mathbf{X}^{(l+1)} = \sigma(\tilde{A}\mathbf{X}^{(l)}\mathbf{W}^{(l)}),$$

其中，归一化邻接矩阵 $\tilde{A}$ 等于 $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$。$\hat{A} = A + I$ 是邻接矩阵加上自邻接，而 $D_{ii} = \sum_{j=0} \hat{A}_{ij}$ 是其对角出度矩阵。$X（l）\in \mathbb{R}^{N \times d}$ 表示第 l 层的节点特征嵌入，而 $W（l）$ 是可学习参数或权重，σ 表示非线性激活函数（例如 ReLU）。Bi-GCN [18] 由两个这样的图卷积网络卷积层组成。

SAGEConvSAGEConv [23] 通过引入均值聚合器来增强图卷积层，因为均值聚合器是局部谱卷积 [7]的线性近似。它还通过添加跳跃连接扩展了该层，以模拟卷积聚合器中拼接上一层节点表示的操作。SAGEConv层定义如下：

$$\mathbf{X}_i^{(l+1)} = \mathbf{X}_i^{(l)}\mathbf{W}_1^{(l)} + \text{mean}_{j \in \mathcal{N}(i)}\mathbf{X}_j^{(l)} \cdot \mathbf{W}_2^{(l)}$$

GraphConv GraphConv [24] 与 SAGEConv 类似，但使用普通求和聚合器：

$$\mathbf{X}_i^{(l+1)} = \mathbf{X}_i^{(l)}\mathbf{W}_1^{(l)} + \sum_{j \in \mathcal{N}(i)} \mathbf{X}_j^{(l)}\mathbf{W}_2^{(l)}$$

为支持在大型图上进行学习，GraphSAGE [23] 提出了一种邻居采样方法，以解决大型图训练中的邻居爆炸问题。Bi-GraphSAGE [18] 由两个这样的 SAGEConv层组成。GraphSAINT [25, 26] 引入了一种基于图采样的归纳学习方法。本文所指的 Bi-GraphSAINT [18]模型由两个 GraphConv 层和一个全连接层组成。

2.2 二值神经网络

两个 0/1 位向量之间的位点积可以按如下方式计算：

$$c = popc(a_{(b)} \ \& \ b_{(b)}),$$

其中，& 表示逻辑与运算，$popc()$ 是用于统计 1 比特数量的人口计数函数。

在二值神经网络（BNN）中，[13–16], 二值化函数通常定义如下：

$$x_{(b)} = sign(x) \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

+1/-1 位向量的位点积运算可通过以下方式计算：

$$c = n - 2 \times popc(a_{(b)} \oplus b_{(b)}) = 2 \times popc(a_{(b)} \odot b_{(b)}) - n$$

其中，$\oplus$ 是异或（XOR），$\odot$ 是同或（XNOR），n 是 $a_{(b)}$ 和 $b_{(b)}$ 的位宽 。

2.3 位操作内建函数

除了常见的逻辑与（&）、或（|）、异或（∧）、取反（¬）、左移位（≪）和右移位（≫）之外，GPU 还配备了若干整数内在函数，可用于高效的位操作。__popc() 和 __popcll() 可对单个 32 位或 64 位无符号整数快速进行位累加。__shfl_sync() 用于线程束 [27] 中线程间的寄存器数据交换。通过指定要交换数据的线程通道，该内在函数会从指定线程通道接收寄存器数据。__ballot_sync() 是一种线程束投票内建函数，用于线程束中线程间的寄存器数据交换。假设线程束中的所有线程均处于活动状态，它将返回一个经过位掩码处理的 32 位无符号整数，表示每个线程中谓词参数的布尔求值结果。这相当于将位列顺时针转置为位行，可用于借助一个线程束中的线程快速对全精度值进行二值化。__brev() 和 __brevll() 分别反转 32 位和 64 位无符号整数的位序。与 __ballot_sync() 配合使用时，它们有助于将位列逆时针旋转 90。度，变为位行 [28]。我们高效二值图神经网络的实现基于这些内在函数。其他厂商的 GPU[29], 也具备类似的内在函数，因此我们的 BitGNN 可移植到其他 GPU平台。

3 BitGNN

我们的 BitGNN 设计旨在实现以下目标：(i) 高效：使用 BitGNN 的图神经网络应能获得显著更高的速度；(ii) 准确：使用 BitGNN 的二值图神经网络应能达到与以往方法实现的二值图神经网络相同的准确率；(iii) 易于使用：借助 BitGNN，用户应能轻松地将图神经网络转换为二值图神经网络；(iv) 灵活调优：不同的图神经网络可能需要不同的精度-速度目标；借助 BitGNN，用户应能轻松调节这种权衡。

本节介绍我们的设计。BitGNN 的设计由一系列新颖的抽象、表示、算法和优化组成。最终成果是一个库及一套实用工具，用户可以利用它们轻松构建和调优高效的二值图神经网络。

图神经网络的两个关键组成部分是操作和张量。相应地，我们创建 BitGNN 的工作围绕三个关键方面展开。

研究问题：(R1) 常见二值图神经网络操作应采用怎样的抽象？(R2) 应如何表示二值张量？(R3) 如何将它们组装成高效且可直接使用的库？

3.1 R1：常见二值图神经网络操作应采用什么抽象？

图神经网络有许多种类，而其二值化形式可能包含更多复杂性和变体，这取决于对图神经网络的哪一部分进行二值化，以实现良好的精度与速度权衡。为了使 BitGNN 能够轻松应用于各种图神经网络，将常见的二值图神经网络操作抽象为一些构建模块非常重要。这种抽象的结果必须涵盖各种二值图神经网络中最重要的操作，同时支持它们在二值化方面的不同需求。据我们所知，尚无先前工作对二值图神经网络操作进行系统性抽象。

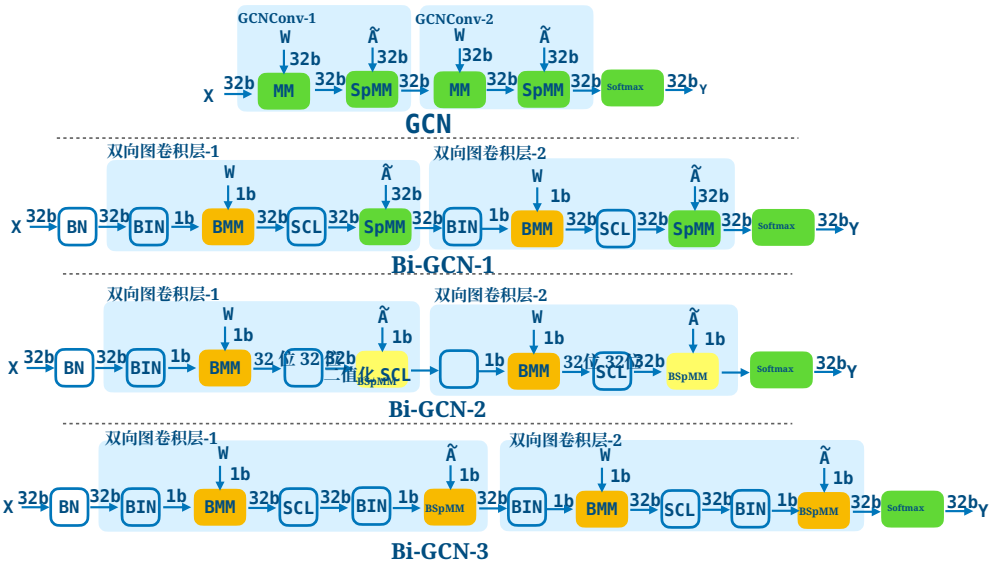
3.1.1 复杂性。 为了更深入地理解设计二值图神经网络（GNN）操作所涉及的复杂性，本节以图卷积网络（GCN）为例，进一步探讨二值化过程。

图1上方的图展示了原始 GCN [7]。GCN 的前向传播包括两个 GCNConv层，随后是一个 Softmax。每个 GCNConv层首先执行矩阵乘法（MM），将输入激活值与权重矩阵相乘。然后，将每个图节点处的结果与其邻居节点的结果进行聚合，这一过程通过与该图的邻接矩阵相乘来完成。由于邻接矩阵通常是稀疏的，第二次矩阵乘法可以使用稀疏矩阵乘法（SpMM）。

图1 中原始图卷积网络下方展示了三种图卷积网络的二值化形式。第一种 Bi-GCN-1 基于先前的一项工作 [18]。它通过离线对权重矩阵进行二值化，将第一个 MM 替换为二值矩阵乘法（BMM）。它还包含三项额外操作：批归一化（BN）、BMM 前的张量二值化（BIN），以及 BMM 后的缩放操作（SCL），用于恢复结果的尺度。第二个 MM 也被替换为 BMM，并分别在 BMM 前后插入 BIN 和 SCL 操作。

Bi-GCN-1 的二值化形式仅关注原始图卷积网络中两个 MM 操作的二值化。然而，图卷积网络中的其他操作也可以进行二值化，且二值化程度可能各不相同。例如，图1 中的 Bi-GCN-2 展示了一种变体，其中两个稀疏矩阵乘法（SpMM）操作进行了“半”二值化——也就是说，邻接矩阵经过了二值化，但激活图没有。相比之下，图1 中的 Bi-GCN-3 展示了两个 SpMM 操作完全二值化的情形，并包含更多辅助操作。这三种变体各有优缺点：随着二值化程度提高，加速比可能更大，但精度损失的风险也会增加。显然，除这三种变体外，还可以有许多其他变体，每种都对应于以特定程度进行二值化的操作组合。

当图神经网络包含额外操作时，设计二值图神经网络操作会变得更加复杂，如顶部所示的 SAGE 和 SAINT 原始架构所示。



Fig图1：原始全精度图卷积网络 [7], 与若干二值图卷积网络变体的示意图。

图 2。我们对众多常见图神经网络及其二值化形式进行了分析，并得出了以下几点观察：

(i) 对于给定的图神经网络，可能存在多种二值化形式，每种形式都涉及大量额外操作。

(ii) 简单的一对一映射并不足够。经过各种二值化处理后，原始图神经网络中的单个操作可能会被不同的操作序列取代。因此，仅为某个原始操作的二值化形式创建一种抽象是不够的。

(iii) 有一组操作构成了二值图神经网络的核心操作。例如，图1所示的各种变体尽管在具体连接方式和架构上有所不同，但都由批归一化、二值化、BMM、SCL、稀疏矩阵乘法和 Softmax 组成。这种模式也出现在其他图神经网络中，因为大多数图神经网络都以矩阵乘法和稀疏矩阵乘法为核心操作。

3.1.2 BitGNN 两级抽象。

基于我们的观察，我们提出了两层 BitGNN 抽象，并按照以下原则进行设计：

(1) 覆盖性：该抽象必须涵盖常见图神经网络中最耗时的核心部分。

(2) 灵活性：该抽象应满足不同精度与速度权衡的需求。

(3) 效率：该抽象应充分考虑对计算效率的影响。

(4) 易于应用：该抽象应便于在图神经网络开发中采用，使图神经网络能够轻松改造为二值图神经网络。此外，它还应支持轻松调整二值化过程，以实现不同的精度-速度目标。

图 2 底部展示了该抽象的核心组成部分。底层函数提供了主要的构建模块，而高层函数则提供了可直接替换图神经网络中用于二值化的组件的选项。前者更加灵活，后者则更易于使用。

底层函数。 底层函数分为三类。第一类专注于矩阵乘法（MM）的二值化。在以往二值图神经网络的实现中，每一层都要对激活张量重新进行二值化，这是性能瓶颈之一。我们的函数设计考虑到了这一点。如图 2 底部所示，我们纳入了七种 BMM 变体，每种对应输入激活值、权重矩阵和输出精确率的一种不同组合。这些精确率决定了所需的辅助操作数（例如，全精度输入在参与乘法前需要进行二值化）。在我们的设计中，这些操作数被纳入 BMM 函数，以避免调用开销和复杂的内核融合需求。

我们的一个发现是，当二值化紧接在 SCL 之后时，SCL 会变得多余，可以将其移除。原因是缩放因子始终为正数2，因此对缩放因子进行逐元素乘法不会影响二值化结果。

第二组底层函数与稀疏矩阵乘法（SpMM）操作有关，这类操作常用于图神经网络中涉及邻接矩阵的乘法 [30]。如图 2 底部所示，我们设计了八种二值化形式。这些变体的设计原因包括与 BMM 情况类似的精度与速度权衡，以及邻接矩阵的特殊属性。在图神经网络工作负载中，两个节点之间的连接可能带有权重，也可能不带权重。前一种情况下，邻接矩阵中只有 0/1 值，用于表示节点连通性。后一种情况下，仍可采用标准方法，通过因子分解对邻接矩阵进行二值化，从而应用 BSpMM。矩阵乘法之后，还会与一个全精度因子分解向量进行乘法运算。

第三组由辅助操作构成，例如用于自连接的加法操作和拼接操作。由于

[2]例如，在 Bi-GCN [18], 中，缩放因子是按行和按列的 L1 归一化值。

二值运算可能会输出不同精确率的结果，因此该组包含若干变体。不过，与其他操作不同，这两种操作的操作数采用混合精度在实践中没有意义，因此我们的设计不包含这些变体。

记法：为方便起见，以下讨论使用三**个字母的后缀**来区分某个运算符的不同变体。例如，BSpMM.FBB 表示 BSpMM：它以全精度（F）矩阵（第一个操作数）和二进制（B）矩阵（第二个操作数）作为输入，并生成一个二进制（B）矩阵。

高层函数。底层函数涵盖了二值图神经网络中最基本的操作，并提供了灵活性，以满足各种需求。为便于采用，我们还设计了一组高层函数，可在图神经网络中轻松替换需要二值化的关键耗时操作。我们的研究表明，尽管图神经网络种类繁多，其最耗时的核心操作都是矩阵乘法（MM）和稀疏矩阵乘法（SpMM）。因此，我们的高层函数围绕这些操作及其组合展开。

高层函数分为两组，每组结合多个底层函数，以便于采用二值图神经网络。第一组针对一种典型情况设计：BSpMM 紧随 BMM 之后执行，如图 2 所示的 GCN。具体变体由 BMM 与稀疏矩阵乘法运算之间的激活张量决定。该张量可以是二值化形式（B），也可以是全精度形式（F）。因此，对于第一个 GCNConv层，我们有 BMM.FBB+BSpMM.BBB 或 BMM.FBF+BSpMM.FBB；对于第二个 GCNConv层，则有 BMM.BBF+BSpMM.FBF 或 BMM.BBB+BSpMM.BBF。第二组针对另一种常见情况设计：在 MM 之后执行某些辅助操作，如图 2 所示的 SAGE 架构。该辅助运算符来自自连接；为此，我们可以使用 ADD.BBF 或 ADD.FFF，来合并第一个 SAGEConv层末尾的激活输出。

图 2 上方展示了如何通过用相应的高层函数替换常见的操作序列，轻松将这三种图神经网络转换为二值图神经网络。延迟较低的底层函数（如 ADD）会与 BMM 或 BSpMM 融合；而高层函数则通过在 BMM 和 BSpMM 之间进行层间融合，并采用协作式内核启动来优化，从而改善线程块同步。此外，高层函数使用户能够轻松调整二值图神经网络，在精度与速度之间进行权衡。例如，用户可以将图 2 所示 GCN.bin 中的两个矩阵乘法-稀疏矩阵乘法替换为高层函数集合中的其他选项。只要前驱模块的输出精确率与其后继模块的输入精确率相匹配，就能保证类型正确性，而准确率和速度则可能有所不同。

总而言之，BitGNN 的高层和低层抽象设计全面覆盖了二值图神经网络架构利用的决策空间。在低层，该设计满足核心操作的各种精度要求，同时最大限度地降低调用开销。在高层，该设计避免不必要的重新二值化，为调整精度与速度权衡提供了灵活性，并支持将图神经网络转换为二值图神经网络的直接替换。

3.2 R2：如何表示二值化张量?

二值化张量的表示方式对二值化图神经网络的内存使用效率和数据访问速度起着至关重要的作用。然而，现有的二值化图神经网络实现（例如 [13–16] 中的实现）并未真正以比特形式存储激活或权重张量，也未考虑以比特形式存储图。

在传统 DNN 中，一些研究探讨了二值张量的表示，并提出了多种表示方式，如 [28, 31] 所示。然而，图神经网络中的两项特殊复杂性需要创新的解决方案。接下来，我们将重点讨论这两项复杂性及我们提出的解决方案。

3.2.1 粒度困境。第一个复杂性与表示的粒度有关。

困境。针对稀疏矩阵，人们提出了许多表示方式，其中一种特别适用于邻接矩阵的方式是基于块的表示 [30]。在这种表示中，矩阵被视为由许多 k×k 块组成，每个块的内容都以稠密格式存储。块级表示采用稀疏格式（例如 CSR），并忽略全零块。

然而，这种表示方式在块大小上面临两难困境。块越小，表示中包含的零就越少，但块之间的归约操作也会越多。较小的块也不利于充分发挥 GPU 的大规模并行能力。

解决方案。为解决块大小带来的困境，我们提出了细粒度表示与动态粗化（*FRDC*）。该方法利用细粒度表示实现较高的空间效率，并通过片上粗化提升时间效率。具体而言，FRDC 使用 4×4 块大小进行表示。选择 4×4 是因为需要使用细粒度表示存储矩阵，同时在计算时采用在线拼接的粗粒度表示。该方法既节省存储和内存空间，也能更好地利用位级并行。

使用较大的块尺寸（如 4×8 或 8×4 ）会使内存用量翻倍，而较小的图块则会增加动态拼接开销。我们的实证结果表明，4×4 是合适的块尺寸。在计算过程中，包含 16 比特的 4×4位块会从 GPU DRAM 传输到共享内存/寄存器。随后，Bit-GNN 内核会在共享内存上将 4×4 位块组装成 32 位对齐的字，并使用字级比特操作内在函数高效处理它们（见第 2.3 节）。采用这种方法，我们既能保留细粒度表示的空间优势，又能实现最大程度的并行化。

3.2.2 数值范围不一致。第二个特殊复杂性源于二值化后张量的数值范围各不相同。

一种常见情况是邻接矩阵与激活图之间的不一致。邻接矩阵使用 1/0 表示边连通性，而基本运算是 0/1 点积。另一方面，二值神经网络激活使用 +1/-1 点积。这种不一致给高效地在二者之间执行乘法带来了挑战。然而，以往的研究并未解决这一问题。

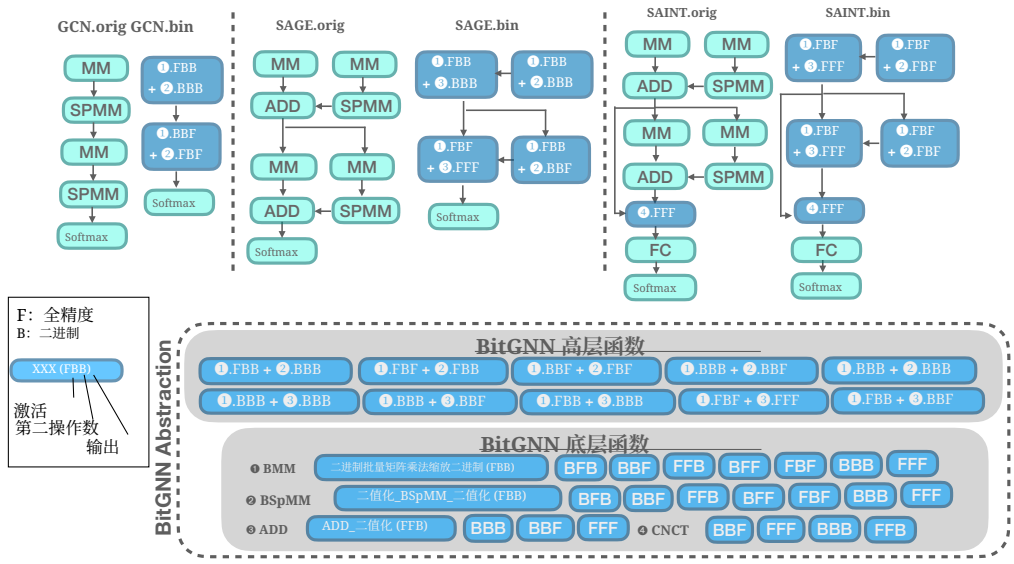


图 2：两层 BitGNN 抽象的核心组件（底部）及其使用示例（顶部）。在底层，BitGNN 支持位运算中的核心内核，包括 BMM（位矩阵乘法）、 BSpMM（位稀疏-稠密矩阵乘法）和 辅助操作。在高层，BitGNN 为二进制或全精度操作数输入/输出提供融合算子。在使用示例中，后缀"orig"和"bin"分别表示图神经网络的原始形式和二进制形式。

在这项工作中，我们提出了三种方法来协调这种不兼容性。设 a 表示邻接矩阵中基于 0/1 的比特向量， b_{org} 表示激活矩阵中基于 +1/-1 的比特向量。 a 和 b_{org} 在内存中都以 0/1 比特表示，但为便于说明，我们引入 b 来表示 b_{org} 在内存中的 0/1 表示，其中 1 对应 +1，0 对应 -1。现在我们介绍计算 a 和 b_{org} 之间点积的三种方法。

(1) 对 a 的非零元素进行 if-else 求值：该方法检查 a 的每一位。如果值为 1，则将 $b(1 \rightarrow 1, 0 \rightarrow -1)$ 中对应的位加到临时和中；否则，跳过 b 中的位值。

(2) $popc(a\&b) - popc(a\&\neg b)$： $popc(a\&b)$ 累加 $a \times b$ 中 1 的总数。 $popc(a\&\neg b)$（$\neg b$ 表示 b 的补码）给出 b_{org} 中 -1 的总数。将两者相减即可得到 a 和 b_{org} 的点积。

(3) $2 \times popc(a\&b) - popc(a)$：将 +1/-1 转换为 1/0，需要先加 1，再除以 2（即右移）。因此，$b = (b_{org} + 1)/2$。由此可得， $b_{org} = 2 \times b - 1$。由于 $popc(a\&b)$ 是 a 与 b 的位点积，我们可以通过 $2 \times popc(a\&b) - popc(a)$ 计算 a 与 b_{org} 的点积。

如果激活值以比特表示，Solution-2 和 Solution-3 比 Solution-1 更高效。然而，如果激活值采用其他形式，Solution-1 可能性能更好。第 3.4 节将进一步讨论如何选择合适的方法。

3.3 R3：如何通过编码实现这一潜力?

第三个研究问题涉及如何有效地实现 BitGNN 的抽象，从而充分发挥张量二进制表示的效率优势。

此前关于二值图神经网络的研究，例如 [18–20], , 仅使用符号函数和 FP32 乘法来模拟二值化，并未在实现的操作中将数值视为比特。

为了解决这一问题，我们提出的方案深入探索了现代 GPU 提供的位操作内建函数，以充分释放 BitGNN 的性能潜力。在 BitGNN 中，核心操作是二值矩阵乘法（BMM）和二值稀疏矩阵乘法（BSpMM），如图 2 所示。这两种操作都有多个变体。虽然以往研究探讨过 BMM，BitGNN 中的 BMM 则建立在先前工作的基础上，同时考虑了不同的精确率以及辅助操作的融合。然而，BSpMM 此前尚未得到研究。因此，下面的讨论将聚焦于 BSpMM。

3.3.1 设计原则。 BitGNN 中使用基于 FRDC 的张量的 BSpMM 内核，遵循以下设计原则：

(1) 基于线程束的工作负载划分：我们将节点的每个工作负载单元划分到一个或多个线程束中，并充分利用 GPU 上快速的线程束内通信 [32] 。

(2) 最大化位级并行：这解决了前文提到的粒度困境，并最大限度地提高边遍历吞吐量（以每秒遍历的边数（TEPS）衡量）。我们确保稀疏图中的小型位块最终以字对齐方式进行操作。

(3) 最大化位张量加载与存储效率：稀疏矩阵乘法通常受内存限制 [33, 34],, 因此我们必须谨慎利用用内存层次结构来操作位张量，以确保最高效率。

3.3.2 实现。 遵循前面提到的三个原则，我们使用 FRDC 方案和位操作内置函数实现 BitGNN 的抽象。下面以 BSpMM.BBB 为例，详细介绍其实现过程。

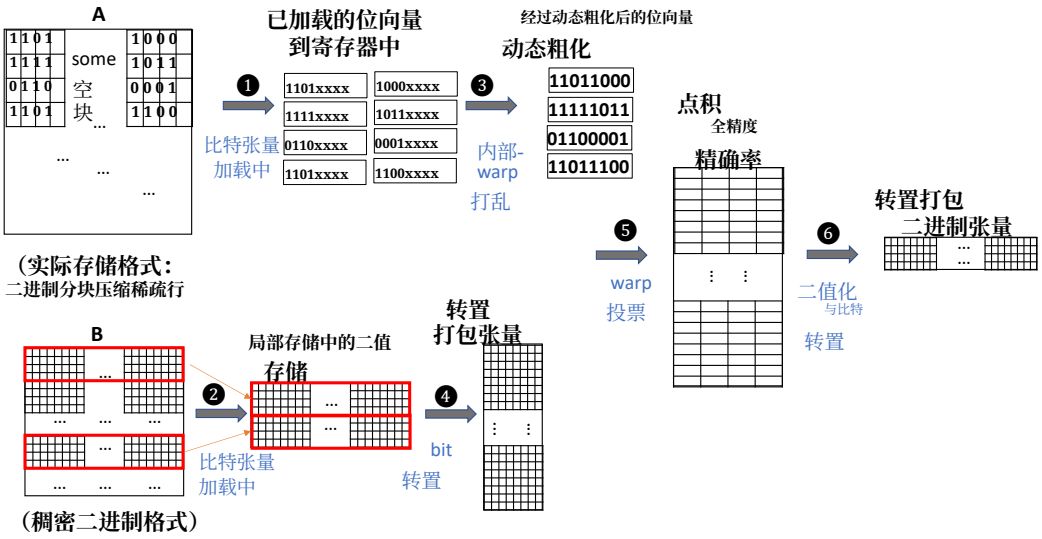


图 3：说明　　使用 FDRC（细粒度表示动态粗化）执行 BSpMM.BBB 的过程　　.

算法 1： BSpMM.BBB

输入： 邻接矩阵：行指针、列索引、位图块、块行标识符 r;
输入激活矩阵$_{(b)}$;
输出： 输出激活矩阵$_{(b)}$
$NT \leftarrow$ 行指针$[r+1]$-行指针$[r]$; $TS \leftarrow \frac{SIMD_unit}{tile_dim}$
对于每个图块集 t_s 其中 $s = 1$ 为 $\lceil \frac{NT}{TS} \rceil$ **执行**
　寄存器 A $\leftarrow$ 位图块$[s]$ for $s = 0$ to TS-1
　寄存器 B $\leftarrow$ 输入激活矩阵$_{(b)}[k]$ 用于 $k =$ 列索引$[s] \times tileDim +$ {0,1,2,...,$tileDim$-1} 和 $s = 0$ 到 TS-1
　寄存器 a $\leftarrow$ a | **线程束洗牌**(A, $s \times tile_dim + laneid$) $\ll$ (4×(TS-1-s)) 对 $s = 0$ 至 TS-1 // 执行位拼接
　共享内存b$[warpid \times \lceil \frac{32}{W} \rceil + k] \leftarrow$ **位反转(投票指令(**(B≫(31-k))&0x1**))** 对 $k = 0$ 至 $SIMD_unit$-1 // 执行位转置
　寄存器c$[n] \leftarrow$ **置位计数(线程束洗牌(**a,n**)**&b$[warpid \times \lceil \frac{32}{W} \rceil + laneid]$)
　- 置位计数(线程束洗牌(a, n)&¬b$[warpid \times \lceil \frac{32}{W} \rceil + laneid]$) 循环 $n = 0$ 至 $tileDim$-1 // 计算位点积
寄存器 rs$[n] \leftarrow$ **位反转(投票指令(**c$[n] \geq 0$**))** 循环 $n = 0$ 至 $tileDim$-1
输出激活矩阵$_{(b)}[r \times tileDim + n] \leftarrow$ rs$[n]$ 循环 $n = 0$ 至 $tileDim$-1

BSpMM.BBB 算法：在算法 1 中，我们展示了采用 FRDC 技术实现 BSpMM.BBB 内核的方法。二值邻接矩阵使用 RowPtr、ColInd 和 BitTiles（BitBlock）[3] 表示。该算法通过图块（块）行维度进行并行化，并展示了图块（块）行 r 的工作负载。

在第 1 行中，每个图块行的图块总数（NT）被划分为若干图块集。图块集的大小（TS）定义为拼接成一个 $SIMD_$单元所需的位行数，等于 $\frac{SIMD_unit}{tile\ dimension}$。例如，当邻接矩阵以 4×4 格式存储，且包含 4×4 个非空图块时，一个非空图块包含四个半字节（4 位）。当我们使用 32 作为一个 $SIMD_$单元时，图块集大小（TS）等于 8。因此，在第 2 行中，内核每次迭代会处理 8 个位图块。一个图块集（TS）是一次迭代的处理单元，而每个图块行的工作负载需要 $\lceil \frac{NT}{TS} \rceil$ 次迭代才能完成。

在这种情况下，邻接矩阵（A 见图 3）和输入激活值（B 见图 3）都是二进制的，因此生成的输出也应为二进制。如图 3 所示，该算法包括以下步骤：

步骤 ❶ & ❷：基于线程束的工作负载划分与加载。为高效利用 GPU 架构，每个由 32 个线程组成的线程束都会映射到 4×4 位图块中一行的计算（即以比特表示图时的 4 个节点）。在线程束内，位图块行会进一步划分为多个图块组，每个图块组包含 8 个位块。例如，如果有十个位图块，则会有两个图块组。第一个图块组包含该 4×4 图块行中的第 0 至第 7 个图块，第二个图块组包含第 8 和第 9 个图块，其余 8 个槽位则以零填充。

接下来，每次迭代都会按如下方式处理一个图块组：线程束中的 32 个线程协同将稀疏比特矩阵 A 中 8 个比特图块的 32 个无符号字符型[4] 从全局内存加载到寄存器中。然后，32 个线程将与已加载图块对应的输入激活值B 的 8 个片段 A 从全局内存加载到共享内存中，形成连续布局，每个字都作为一个 32 位向量使用。

步骤 ❸ 通过比特拼接即时组装。在此步骤中，会执行动态粗化，以确保 A 的比特图块按 32 比特对齐。第 0 至第 3 个线程使用线程束内通信内建函数（shfl_sync()）和位移操作，将每个节点的邻居（以位向量形式）快速拼接成一个 32 比特无符号位向量（图 3 的示意图中仅显示了 8 比特）。

步骤 ❹ 位转置，实现按列合并访问。此步骤将 B 的列优先打包方式转置为行优先打包方式。这样，在与 A 计算位点积时，位

[3]我们在段落中交替使用“图块”和“块”这两个术语。

[4]在我们的原始实现中，我们将 4 位单元存储为无符号字符型（1 字节）。在较新的 GPU 架构中，也可以使用更小的表示形式（即 4 位整数）来存储它们。

就可以按位列读取，从而充分发挥位级并行的优势。通过对 B 的位向量进行位移和掩码操作来完成转置。随后，使用线程束内投票内在函数 ballot_sync() 对结果进行评估和收集。最后，使用 brev() 内在函数完成转置。

步骤 三值位点积。在此步骤中，算法计算 A 的 0/1 位向量与 B 的 +1/-1 位向量之间的位点积。（“三值”指 0/1/-1。）循环迭代 4 次（每个节点一次），处理来自 A 的一个 32 比特位向量。32 个线程各自负责计算位点积的 1/32。乘积以全精度值的形式存储在寄存器中。

步骤 位张量存储。在此步骤中，通过逐元素判断乘积是否大于或等于 0，将临时全精度乘积打包为位格式，这就是二进制激活张量的二值化过程。线程束中的 32 个线程使用线程束内投票内在函数（即 ballot_sync()）协同处理全精度值。随后，brev() 内在函数将结果逆时针转置，以完成位向量打包。输出为激活映射的 32 比特无符号位向量。

其他变体 其他 BSpMM 变体的实现采用类似的方法，大多数计算使用位级内置函数，但处理不同精确率的方式有所不同。例如，BSpMM.FBB 以不同的方式加载激活张量。在步骤 （位张量加载）中，需要 4 个线程束协同加载所有全精度值。每个线程束中的线程负责加载 32 个邻居节点的特征向量中的一个维度。因此，不需要执行步骤 （为列方向合并访问进行位转置）。更多详情请参阅补充材料 [35]。

3.4 调优实用工具及其他实现细节

为了启动包含多个内核的端到端模型，我们使用 cudaLaunchCooperativeKernel() 来实现特定线程块集合之间的全局同步 [36]。我们首先使用 cudaOccupancyMaxActiveBlocksPerMultiprocessor() 计算每个流式多处理器（SM）能够容纳的最大块数，然后为每个线程块使用 1024 个线程，以允许每个 SM 上尽可能多的线程束执行。这确保我们能够充分利用 GPU 上可用的并行性，并实现最佳性能。

BitGNN 的调优实用工具提供了优化二值图神经网络的便捷方法。如前文所述，BitGNN 提供了一系列变体，以满足图神经网络输入、权重和输出的不同精度要求。调优实用工具支持自动调优运行，以其他变体替换 BitGNN 函数调用并测量其性能。通过确保前驱模块的输出精度与其后继模块操作的输入精度相匹配，可以保证类型正确性。此外，调优实用工具还便于选择最佳方案，以解决第 3.2.2 节讨论的二进制值范围不一致问题。还可以开发预测器来预先判断针对特定的图神经网络（GNN）和图，选择最优变体。类似的预测器已在其他领域得到广泛研究，例如稀疏矩阵存储格式 [37]。

4 评估

本节报告 BitGNN 的性能。我们重点关注以下问题：(i) 它能为图神经网络推理带来多大的加速效果？(ii) 它能节省多少内存空间？(iii) 使用 BitGNN 时，精度与速度之间存在怎样的权衡？

4.1 方法论

GPU 环境： 我们在三款不同架构代际的 NVIDIA GPU 上评估 BitGNN 的性能：GTX 1080 （帕斯卡）、 TITAN V （伏特）和 RTX 3060Ti（Ampere）。其特性总结于表 1 中。CUDA 版本为 11.0。所有实验均采用 10 次重复执行的平均值。 **数据集：** 表 2 展示了评估中使用的图。这些图常用于现有图神经网络研究 [23, 25, 26, 38–41] ，也用于先前的二值图神经网络工作 Bi-GCN [18]，因此可以进行直接比较。

要比较的版本： 我们将不同精度的 BitGNN 版本与 Bi-GCN [18] 进行比较。作为当前最先进的二值图神经网络代表——这些方法都侧重于二值化算法，而非优化执行，Bi-GCN 对图神经网络中的操作进行二值化，但仍使用值的全精度表示。同时，为提供参考基准，我们报告了未进行二值化的原始图神经网络的性能和准确率。原始版本和 Bi-GCN 版本均在 PyG [22] 中实现。

图神经网络： 我们在实验中使用了四种图神经网络：传导式图卷积网络 [7]，、归纳式图卷积网络 [23]，、GraphSAGE [23], 和 GraphSAINT [25, 26]。选择这些图神经网络是因为它们最为流行，也是此前我们进行比较的二值图神经网络研究 [18] 所使用的图神经网络。

4.2 结果与分析

表 3、表 4 和表 5 报告了基线方法和 BitGNN 各版本的速度（推理时）、准确率和空间占用。

在深入讨论结果之前，值得一提的是，图神经网络学习分为两种：传导式学习和归纳式学习。在传导式学习中，整个数据图会在学习过程中被观察到，其中一些节点有标签，另一些则没有。图神经网络算法尝试通过学习整个图结构和那些有标签的节点，迭代地优化无标签节点的标签。在归纳式学习中，图神经网络算法从一些采样图中学习，然后尝试预测其他采样图中节点的标签。

在先前研究使用的四种图神经网络中， [7, 23, 25, 26],传导式-*GCN* 用于传导式学习，而另外三种图神经网络用于归纳式学习。由于大型图难以按照传导式图神经网络的要求整体加载并在内存中处理，先前工作仅将传导式-*GCN* 应用于小型数据集（Cora、PubMed、CiteSeer）。对于大型数据集（ Flickr 和 Reddit），先前工作采用三种归纳式图神经网络。为实现直接对比，我们的实验遵循相同做法。

表 1：用于评估的 GPU。Arch 指 GPU 架构代际。CC 指计算能力。SMs 指 GPU 中流式多处理器的数量。Thrds 指线程数。Shared 指共享内存大小。Reg 指寄存器数量。注意，“/Block”和“/Thrd”分别表示每个线程块和每个线程可使用的最大资源。

GPU	架构	CC	流式多处理器	DRAM	内存带宽	L1缓存 每个 SM 的大小	L2缓存 Size	线程束/SM	块/SM	线程/SM	共享/SM	共享/块	Reg/SM	Reg/块	Reg/线程
GTX 1080	帕斯卡	6.0	20	8GB	320GB/s	48KB	4096KB	64	32	2048	64KB	48KB	64K	64K	255
TITAN V	伏特	7.0	80	12GB	653GB/s	96KB	4608KB	64	32	2048	96KB	96KB	64K	64K	255
RTX 3060 Ti	Ampere	8.0	38	8GB	448GB/s	128KB	4096KB	64	32	2048	164KB	164KB	64K	64K	255

表 2：用于评估的图数据集。

数据集	#节点	#边	#特征	#类别
Cora	2,708	13,264	1,433	7
PubMed	19,717	108,356	500	3
Citeseer	3,327	12,431	3,703	6
Flickr	89,250	899,756	500	7
Reddit	232,965	114,615,892	602	41

表 3：传导式图卷积网络的评估结果。FP32 是原始图卷积网络 [7]。“FP32 (S)”指采用最大批次大小的 PyG 散射-收集抽象。“FP32 (T)”指 PyG 稀疏矩阵乘法张量抽象。Bi-GCN [18] 是当前最先进的二值图卷积网络实现。我们比较了在矩阵乘法操作中同时将激活值和权重二值化的版本。“我们的方法（完整）”指 全精度聚合。“我们的方法（二值化）”指使用 MM.FBB + BSpMM.BBB 作为第一层、使用 MM.BBF + BSpMM.FBF 作为第二层的 二值聚合 版本。“峰值内存”指推理期间的峰值内存使用量。“准确率”指准确率百分比（%）。

数据集	模型	Peak 内存（B）	Acc (%)	端到端时间（毫秒） GTX1080	TitanV	RTX3060Ti
Cora	FP32(S) [7]	16.73M	81.4±0.4	1.25±0.01	0.91±0.03	1.00±0.01
	FP32(T) [7]	16.73M	81.4±0.4	1.34±0.02	1.15±0.05	1.16±0.03
	Bi-GCN [18]	16.73M	81.2±0.8	1.84±0.04	1.91±0.02	1.57±0.02
	我们的方法（完整）	1.37M	81.2±0.8	0.51±0.00	0.37±0.07	0.38±0.02
	我们的方法（二值化）	0.73M	81.2±1.0	0.32±0.04	0.28±0.08	0.24±0.04
PubMed	FP32（S）	48.71M	79.0±0.3	2.86±0.04	1.51±0.02	1.90±0.02
	FP32 (T)	48.71M	79.0±0.3	2.26±0.01	1.49±0.02	1.77±0.01
	Bi-GCN	48.71M	78.2±1.0	2.81±0.07	2.23±0.09	1.52±0.01
	我们的方法（完整）	7.31M	78.2±1.0	2.29±0.05	0.84±0.05	1.14±0.00
	我们的方法（二值化）	2.65M	78.1±1.1	0.74±0.04	0.33±0.05	0.44±0.01
CiteSeer	FP32 (S)	49.78M	70.9±0.5	2.60±0.03	1.43±0.04	1.74±0.01
	FP32（T）	49.78M	70.9±0.5	2.70±0.01	1.56±0.01	1.88±0.05
	Bi-GCN	49.78M	68.8±0.9	2.44±0.00	2.22±0.04	1.33±0.09
	我们的方法（完整）	2.56M	68.8±0.9	0.80±0.04	0.58±0.02	0.50±0.03
	我们的方法（二值化）	1.77M	68.7±0.4	0.63±0.03	0.46±0.01	0.42±0.00

所提出的灵活编程抽象（第 3.1.2 节）能够轻松促进用户开发二进制网络。借助它，我们调整了各种二值化精确率，并找出了一些表现相对较好的模型变体：全精度聚合 （与 Bi-GCN [18] 相同）和二值聚合 （第一层使用 MM.FBB + BSpMM.BBB，第二层使用 MM.BBF + BSpMM.FBF）。它们在结果表中分别表示为我们的方法（完整）和我们的方法（二值化）。

传导式场景： 根据表 3，我们得到以下观察结果。(i) BitGNN 的两个版本都能保留大部分准确率。在三个数据集上，平均准确率的最大损失分别为 0.2%、0.9% 和 2.2%。最大精度损失（2.2%）出现在 *CiteSeer* 上，因为边权重归一化的影响更为显著。(ii) 两个版本都能显著降低内存用量，我们的方法（完整）可减少 7–19 倍，我们的方法（二值化）可减少 18–28 倍。(iii) 两个版本在不同 GPU 型号上都持续实现显著加速：我们的方法（完整）可加速 1.2–5 倍，我们的方法（二值化）可加速 4–7 倍。(iv) 由于先前的二值化图神经网络（Bi-GCN）为二值化值使用的是一个字，而非单个比特，因此与原始全精度图神经网络相比，它既不能节省空间，也不能缩短运行时间。(v) 我们更完整的二值化版本（我们的方法（二值化））相比采用全精度聚合的版本（我们的方法（完整）），额外节省了 2–3 倍的空间，并额外实现了 1.5–3 倍的加速，而额外的精度损失仅为 0.1%。(vi) 这些优势在各数据集上基本一致。一个例外是 我们的方法（完整）在 PubMed 上的表现：在 GTX1080 和 RTX3060Ti 上，我们的方法（完整）相较原始版本仅实现了适度加速，而我们的方法（二值化）仍实现了显著加速。我们认为，这是因为节点数量相对较多，导致第一层中 BSpMM.BBB 与 BSpMM.FBB 之间的性能差距较大。

归纳式场景： 表 4 和表 5 报告了两个大型数据集上三种归纳式图神经网络的结果。在空间和时间方面，收益同样显著，但与传导式场景相比存在一些差异。我们注意到以下几点。(i) 对于归纳式学习，先前工作 [23] 使用微平均 F1 分数而非准确率来衡量推理质量。微平均 F1 分数是精确率和召回率的综合指标 [23, 42]。二值化再次保持了大部分质量，仅造成 0.5–2.1% 的准确率损失。(ii) 内存空间节省为 1.72–16 倍。节省幅度不及传导式场景。原因在于，我们的大部分空间节省来自邻接矩阵和节点特征的位表示。但稀疏表示还包含非空块的索引。由于大多数非空块仅包含一个非零值，与默认的压缩稀疏行格式相比，我们在索引数据结构上的节省并不多。对于小型图，索引数据结构仅占整体表示的一小部分；但对于大型图，这一比例会大得多。不过，节省仍然十分可观。(iii) 我们的版本所实现的加速比更加显著，范围从 2 倍到 125 倍。原因在于，随着图规模增大，可节省的时间空间也随之增加。另一个值得一提的好处是，由于内存空间占用大幅降低，新方法可以将整个图加载到内存中，并在一次运行中生成所有采样子图的推理结果；相比逐个处理子图，这可以进一步节省时间。

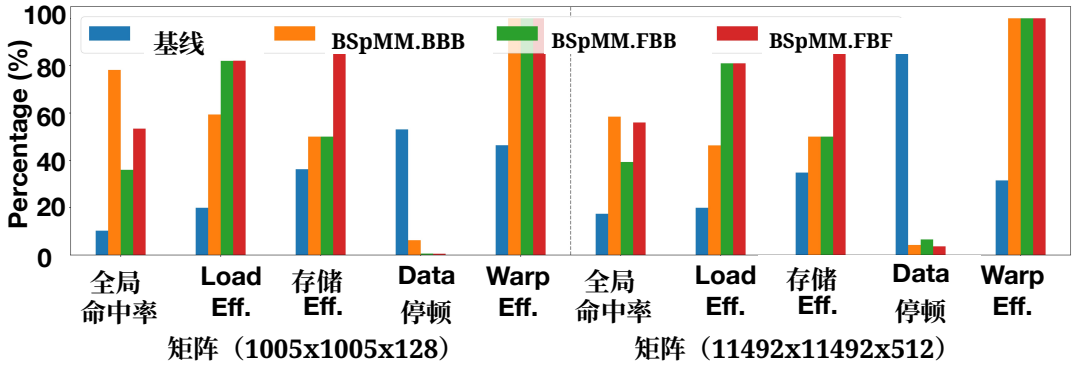


图4：在 Nvidia GTX1080 GPU 上对两对稀疏矩阵进行乘法运算的性能分析结果。表4：归纳式图神经网络（归纳式图卷积网络和 GraphSAGE）的评估结果 [23]。微平均 F1 是衡量归纳式图神经网络准确率的指标。Flicker 的隐藏层维度 为 256；Reddit 的为 128。为避免内存溢出错误，FP32 和 Bi-GCN 推理在不同 GPU 上均设置为最大批次大小。

数据集	模型	归纳式图卷积网络 Peak 内存 (B)	F1-微	端到端时间 (s) GTX1080	TitanV	RTX3060Ti	GraphSAGE Peak 内存（B）	F1-微型	端到端时间 (s) GTX1080	TitanV	RTX3060Ti
Flickr	FP32 (S) [23]	358.07M	50.9±0.3	0.36±0.005	0.33±0.007	0.32±0.009	448.37M	50.9±1.0	0.36±0.006	0.32±0.004	0.32±0.008
	FP32 (T) [23]	358.07M	50.9±0.3	0.045±0.000	0.035±0.000	0.03±0.000	448.37M	50.9±1.0	0.045±0.001	0.035±0.000	0.034±0.000
	Bi-GCN [18]	358.07M	50.2±0.4	0.36±0.007	0.34±0.010	0.34±0.011	448.37M	50.2±0.4	0.36±0.008	0.33±0.003	0.33±0.002
	我们的方法（完整）	107.07M	50.2±0.4	0.023±0.01	0.016±0.000	0.02±0.003	196.9M	50.2±0.4	0.038±0.007	0.027±0.00	0.03±0.006
	我们的方法（二值化）	22.38M	49.9±0.3	0.006±0.004	0.002±0.000	0.005±0.000	27.52M	50.8±0.1	0.008±0.008	0.003±0.00	0.006±0.001
Reddit	FP32（S）	1.67G	93.3±0.0	49.01±0.25	38.19±0.10	38.70±0.28	1.81G	95.3±0.1	49.27±0.18	37.95±0.35	38.29±0.22
	FP32 (T)	1.67G	93.3±0.0	0.86±0.02	0.45±0.001	0.52±0.007	1.81G	95.3±0.1	0.92±0.001	0.51±0.001	0.52±0.002
	Bi-GCN	1.67G	93.1±0.2	49.30±0.28	38.22±0.10	38.97±0.34	1.81G	95.3±0.1	49.42±0.21	38.05±0.56	39.03±0.11
	我们的方法（完整）	1.02G	93.1±0.2	0.82±0.02	0.38±0.08	0.45±0.03	1.17G	95.3±0.1	0.90±0.003	0.45±0.03	0.53±0.04
	我们的方法（二值化）	943.77M	92.8±0.1	0.49±0.11	0.15±0.007	0.27±0.12	983.77M	91.2±1.2	0.53±0.05	0.20±0.009	0.29±0.05

表5：归纳式图神经网络的评估结果 — 图- SAINT [25, 26]。Flicker 的隐藏层维度 为 256；Reddit 的为 128。

数据集	模型	Peak 内存（B）	F1-微型	端到端时间（秒） GTX1080	TitanV	RTX3060Ti
Flickr	FP32（S） [25, 26]	623.72M	51.1±0.1	0.39±0.004	0.36±0.003	0.34±0.006
	FP32 (T) [25, 26]	623.72M	51.1±0.1	0.05±0.000	0.03±0.000	0.04±0.000
	Bi-GCN [18]	623.72M	50.8±0.2	0.05±0.003	0.04±0.000	0.04±0.001
	我们的方法（完整）	371.83M	50.8±0.2	0.15±0.004	0.07±0.00	0.09±0.001
	我们的方法（二值化）	202.39M	49.6±0.8	0.023±0.007	0.012±0.00	0.016±0.001
Reddit	FP32 (S)	2.04G	96.6±0.1	48.79±0.23	38.41±0.32	39.77±0.22
	FP32（T）	2.04G	96.6±0.1	1.25±0.005	0.60±0.001	0.69±0.003
	Bi-GCN	2.04G	95.7±0.1	1.27±0.002	0.61±0.003	0.73±0.011
	我们的方法（完整）	1.39G	95.7±0.1	0.92±0.46	0.66±0.08	0.74±0.08
	我们的方法（二值化）	1.18G	94.5±0.6	0.45±0.16	0.24±0.003	0.33±0.09

此外，表 4 显示，GraphSAGE- Reddit 上的准确率损失为 4.1%。我们将其归因于经过原始训练的二值化 GraphSAGE 的敏感性，以及所使用的较小隐藏层维度。尽管 Reddit 图明显大于 Flickr，但其隐藏节点数最少仅为 128 个。将隐藏节点数增加到 256 和 512 后，准确率损失会减小（~0.67%、 ~0.9%）。在 Flickr 上，准确率甚至提高了 0.6%。图神经网络推理准确率并不稳定，尤其是在大型图上；这并非图神经网络所独有的现象。研究表明，压缩深度神经网络有时可以获得更高的准确率高于原始模型。准确率略有提升，是因为在第一个图卷积网络层中采用了二值聚合近似。

表5报告了 GraphSAINT 上的结果。对于这个图神经网络，两个图的模式（大多数非零图块中仅有一个非零元素）有利于基于张量内核的实现。我们的方法（二值化）在不同架构和数据集上仍持续实现显著的加速比。这得益于我们的实现只加载图块中对应于非零元素的元素。空间节省仍然十分可观。

硬件性能分析： 使用 GPU 厂商的性能分析工具 NSight，我们发现，性能提升主要源于二值化改善了内存利用率。图 4 展示了我们三个 BSpMM 内核的性能分析结果，并与基线进行比较——基线是在计算两对具有代表性的矩阵的乘法时，Torch Sparse 中默认的稀疏矩阵乘法内核；测量在 Nvidia GTX1080 GPU 上进行。具体而言，我们观察到，全局加载/存储效率（从 19% 提升至 80%）和 L1/Tex 缓存命中率（从 10% 提升至 53-78%）均显著提高。此外，数据请求停顿从 53-89% 降至 10% 以下后，线程束执行效率从 31-46% 大幅提升至近 100%。

讨论： 关于需要执行的操作总数，尽管 BitGNN 将 32 个元素的乘加操作聚合为针对一个 32 位操作数的单次位点积操作，但实际的操作数量减少幅度

由于额外的索引开销而不到 32 倍。尽管如此，由于 SpMM 通常受内存限制，二值化仍能显著提升性能。

潜在影响： Bit-GNN 带来的空间和时间缩减不仅提高了服务器图形处理器上的计算效率，还为处理大型图（无法装入内存）以及在资源受限的设备上使用图神经网络开辟了新的机会。量化的（非图神经网络）深度神经网络已经在资源受限的嵌入式场景中的许多领域得到应用，包括自动驾驶 [43], 智慧农业 [44], 新冠疫情口罩检测 [45], 3D 目标检测 [46] 和图像处理 [47]。这主要归因于简化的逻辑 [48], 低能耗 [49], 低硬件要求以及鲁棒性 [50]。然而，许多场景中的输入数据是非欧几里得的，或者使用图能够更好地表示，例如分子 [51], 交通流 [52], 电网 [53], 等。在这些场景中，二值图神经网络作为二值神经网络的替代方案，可以在特征提取和实时预测中发挥重要作用。

5 相关工作

二值图神经网络。 二值图神经网络近年来受到关注，被视为在机器学习社区中将量化应用于图学习工作负载的一种有前景的方法。尽管已有多项研究提出了图神经网络二值化技术和/或二值图神经网络模型 [18–21], ，但它们都未能展现二值化所能带来的全部性能优势，而这正是 BitGNN 的用武之地。下面，我们总结一些现有的二值图神经网络研究。

二值图卷积网络（Bi-GCN） [18] 提出了一种新颖的基于二值梯度近似的反向传播技术，用于有效训练二值图卷积网络。该方法对网络参数和节点特征进行二值化，同时将精度损失降至最低。Bi-GCN 已应用于传导式学习图神经网络（例如 GCN [7]）和归纳式学习图神经网络（例如归纳式图卷积网络、GraphSAGE [23],和 GraphSAINT [25, 26]）。另一种方法是，Bahri 等人 [19] 采用知识蒸馏和多阶段训练，对动态图卷积神经网络（DGCNN） [5]中的边卷积进行二值化。Wang 等人 [20] 提出了一种名为 BGN 的新型二值化图嵌入方法，用于对自适应采样（AS-GCN） [54] 和图注意力网络（GAT） [55]的参数及预激活值进行二值化。在反向传播阶段，他们采用两种常用的无偏梯度估计器——直通估计器和 REINFORCE 估计器 [56] —— 进行二值近似。除了使用同或和人口计数进行 +1 以及 -1 比特计算外，他们还采用掩码求和和平衡函数来改进二值化过程。元聚合器 [21] 引入了两种聚合器——贪心 Gumbel 聚合器（GNA）和可适配混合聚合器（ANA）——旨在提高聚合阶段的二值训练准确率。

尽管这些针对图神经网络的二值化技术为探索二值图神经网络的潜力迈出了第一步，但它们只是在算法层面“逻辑上”将图神经网络二值化，其实现仍在 PyTorch 中使用符号函数和矩阵乘法函数，本质上在底层采用全精度张量进行存储，并使用全精度操作进行计算。因此，它们未能展现二值图神经网络的性能潜力。

GPU 上的位操作。 以往研究已在广泛的应用场景中深入探索了在现代 GPU 上利用位操作的潜力。这些场景包括使用卷积神经网络进行图像分类 [14, 28, 31, 57, 58], 基于立体视觉的深度感知 [43, 59], 欺诈检测 [60], 以及图算法 [30]。

Courbariaux 等人 [14] 首次提出在 GPU 上对小型多层感知机使用异或和人口计数运算的概念。Espresso[57] 构建了一个带有 C 和 CUDA 后端的二值卷积神经网络库，提供位打包和位矩阵乘法函数。Khan 等人 [58] 将此前基于通用矩阵乘法的卷积神经网络扩展到桌面和嵌入式 GPU 上的实时车辆分类。BSTC [28]提出了用于 BMM 和二值卷积的基于位块的内核设计，将显著的性能提升扩展到更深层的二值 AlexNet、VGGNet 和 ResNet。StereoBit [43] 和 FastFusion [59] 开发了基于二值神经网络的立体匹配网络，并通过优化实现来减少 GPU 片上存储器占用。它们使用图像对的二值特征描述符和权重构建网络，以异或和人口计数运算取代开销较大的全精度算术运算。Ye 等人 [60] 利用 GPU 内置函数 __ballot_sync()、__match_any_sync()，以及 __popc()，构建欺诈检测流程的标签传播（LP）算法。Feng 等人 [61] 将量化神经网络矩阵乘法分解为多批 BMM 操作，并依靠 Ampere 张量核心进行加速。最后，Bit-GraphBLAS [30] 提出了一种稀疏位存储格式，以及 GraphBLAS 运算符的二进制实现，以加速基于迭代的图算法，适用于图边同质且可表示为二值图的情况。

这些先前的研究表明，在 GPU 上利用比特操作和运算能够带来性能优势。然而，这些研究尚不够全面，无法解决处理二值图神经网络时出现的特定挑战，例如算子重构的需求、恰当的张量表示以及数值乘法的不一致性。

6 结论

BitGNN 是首项全面解决使用位张量及其操作实现二值图神经网络的设计与挑战的工作。我们提出的编程抽象、张量表示和内核设计技术，能够端到端地最大限度地提升二值图神经网络模型的性能。与以往的二值图神经网络实现相比，这些优化和实现使端到端模型性能实现了显著的 8–22 倍加速。

致谢

本材料所述工作得到美国能源部（高级科学计算研究办公室）资助，资助编号为 DE-EE0009357 和 78284；此外，还得到美国国家科学基金会（NSF）资助，资助编号为 CNS-2107068。本材料中表达的任何观点、发现、结论或建议均为作者所持观点，不一定反映 NSF 或美国能源部的看法。

参考文献

[1] Wenqi Fan、Yao Ma、Qing Li、Yuan He、Eric Zhao、Jiliang Tang 和 Dawei Yin。用于社交推荐的图神经网络。载于《万维网大会》，第 417–426 页，2019 年。[2] Manon Réau、Nicolas Renaud、Li C Xue 和 Alexandre MJJ Bonvin。Deeprank-gnn：一种用于学习蛋白质-蛋白质界面模式的图神经网络框架。《生物信息学》，39(1):btac759，2023 年。[3] Justin Gilmer、Samuel S Schoenholz、Patrick F Riley、Oriol Vinyals 和 George E Dahl。用于量子化学的神经消息传递。载于《国际机器学习会议》，第 1263–1272 页。PMLR，2017 年。[4] Hatem Helal、Jesun Firoz、Jenna Bilbrey、Mario Michael Krell、Tom Murray、Ang Li、Sotiris Xantheas 和 Sutanay Choudhury。面向量子化学的基于图神经网络预测模型的极致加速。arXiv 预印本 arXiv:2211.13853，2022 年。[5] Yue Wang、Yongbin Sun、Ziwei Liu、Sanjay E. Sarma、Michael M. Bronstein 和 Justin M. Solomon。用于点云学习的动态图卷积神经网络。《ACM图学汇刊》，38(5)，2019 年 10 月。[6] Jie-Fang Zhang 和 Zhengya Zhang。Point-X：一种感知空间局部性的架构，用于节能型基于图的点云深度学习。载于 MICRO-54：第 54 届 IEEE/ACM微体系结构国际研讨会，第 1078–1090 页，美国纽约，2021 年。计算机协会。[7] Thomas N. Kipf 和 Max Welling。基于图卷积网络的半监督分类。载于国际学习表征会议（ICLR），2017 年。[8] Liang Yao、Chengsheng Mao 和 Yuan Luo。用于文本分类的图卷积网络。载于 AAAI 人工智能会议论文集，第 33 卷，第 7370–7377 页，2019 年。[9] Hongfan Ye、Buqing Cao、Junjie Chen、Jianxun Liu、Yiping Wen 和 Jinjun Chen。一种基于 GCN 的 Web 服务分类方法。载于 2019 IEEE并行与分布式处理及应用国际会议、大数据与云计算、可持续计算与通信、社会计算与网络（ISPA/BDCloud/SocialCom/SustainCom），第 1107–1114 页。IEEE，2019 年。[10] Jiali Liang、Yufan Deng 和 Dan Zeng。一种结合 CNN 与 GCN 的深度神经网络，用于遥感场景分类。《IEEE应用地球观测与遥感精选主题期刊》，13:4325–4338，2020 年。[11] Sheng Wan、Chen Gong、Ping Zhong、Shirui Pan、Guangyu Li 和 Jian Yang。基于上下文感知动态图卷积网络的高光谱图像分类。《IEEE地球科学与遥感汇刊》，59 612，2020 1 597( ): – 。[12] Hao Jiang、Peng Cao、MingYi Xu、Jinzhu Yang 和 Osmar Zaiane。Hi-gcn：一种用于脑网络图嵌入学习与脑部疾病预测的分层图卷积网络。《生物学与医学中的计算机》，127:104096，2020 年。[13] Itay Hubara、Matthieu Courbariaux、Daniel Soudry、Ran El-Yaniv 和 Yoshua Bengio。二值化神经网络。《神经信息处理系统进展》，29，2016 年。[14] Matthieu Courbariaux、Itay Hubara、Daniel Soudry、Ran El-Yaniv 和 Yoshua Bengio。二值化神经网络：将深度神经网络的权重和激活值限制为 +1 或 -1 进行训练。arXiv 预印本 arXiv:1602.02830，2016 年。[15] Mohammad Rastegari、Vicente Ordonez、Joseph Redmon 和 Ali Farhadi。Xnor-net：使用二值化卷积神经网络进行 ImageNet分类。载于欧洲计算机视觉会议，第 525–542 页。施普林格，2016 年。[16] Adrian Bulat 和 Georgios Tzimiropoulos。Xnor-net++：改进的二值神经网络。载于英国机器视觉会议（BMVC）论文集，2019 年。[17] Yichi Zhang、Zhiru Zhang 和 Lukasz Lew。Pokebnn：以二值追求轻量级精度。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 12475–12485 页，2022 年。[18] Junfu Wang、Yunhong Wang、Zhen Yang、Liang Yang 和 Yuanfang Guo。Bi-gcn：二值图卷积网络。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 1561–1570 页，2021 年。[19] Mehdi Bahri、Gaétan Bahl 和 Stefanos Zafeiriou。二值图神经网络。载于 IEEE/CVF计算机视觉与模式识别会议论文集，第 9492–9501 页，2021 年。[20] Hanchen Wang、Defu Lian、Ying Zhang、Lu Qin、Xiangjian He、Yiguang Lin 和 Xuemin Lin。二值化图神经网络。《万维网》，24 848，2021 3 825( ): – 。[21] Yongcheng Jing、Yiding Yang、Xinchao Wang、Mingli Song 和 Dacheng Tao。元聚合器：学习如何聚合以构建 1位图神经网络。载于 IEEE/CVF 国际计算机视觉会议论文集，第 5301–5310 页，2021 年。[22] Matthias Fey 和 Jan E. Lenssen。使用 PyTorch Geometric 进行快速图表示学习。载于 ICLR图与流形表示学习研讨会，2019 年。[23] William L Hamilton、Rex Ying 和 Jure Leskovec。大型图上的归纳式表示学习。载于第 31 届神经信息处理系统国际会议论文集，第 1025–1035 页，2017 年。[24] Christopher Morris、Martin Ritzert、Matthias Fey、William L Hamilton、Jan Eric Lenssen、Gaurav Rattan 和 Martin Grohe。Weisfeiler 和 Leman 走向神经网络：高阶图神经网络。载于 AAAI 人工智能会议论文集，第 33 卷，第 4602–4609 页，2019 年。[25] Hanqing Zeng、Hongkuan Zhou、Ajitesh Srivastava、Rajgopal Kannan 和 Viktor Prasanna。准确、高效且可扩展的图嵌入。载于 2019 年 IEEE 国际并行与分布式处理研讨会（IPDPS），2019 年 5 月。[26] Hanqing Zeng、Hongkuan Zhou、Ajitesh Srivastava、Rajgopal Kannan 和 Viktor Prasanna。GraphSAINT：基于图采样的归纳式学习方法。载于国际学习表征会议，2020 年。[27] Ang Li、Weifeng Liu、Linnan Wang、Kevin Barker 和 Shuaiwen Leon Song。Warp 合并：一种面向 GPU 的新型执行模型。载于 2018 年国际超级计算会议论文集，ICS '18，第 53–64 页，美国纽约州纽约市，2018 年。计算机协会。[28] Ang Li、Tong Geng、Tianqi Wang、Martin Herbordt、Shuaiwen Leon Song 和 Kevin Barker。BSTC：一种用于加速基于比特的近似神经网络的新型二值化软张量核心设计。载于高性能计算、网络、存储与分析国际会议论文集，SC '19，美国纽约州纽约市，2019 年。计算机协会。[29] AMD。HIP 编程指南：https://github.com/radeonopencompute/rocm/blob/rocm-4.5.2/amd_hip_programming_guide.pdf，2022 年。[30] Jou-An Chen、Hsin-Hsuan Sung、Xipeng Shen、Nathan Tallent、Kevin Barker 和 Ang Li。Bit-GraphBLAS：GPU 上以图为中心的处理的比特级优化。载于 2022 年 IEEE 国际并行与分布式处理研讨会（IPDPS），第 515–525 页，2022 年。[31] Ang Li 和 Simon Su。通过 Turing GPU 中的比特张量核心加速二值化神经网络。IEEE 并行与分布式系统汇刊，32(7):1878–1891，2020 年。[32] Eli Ben-Sasson、Matan Hamilis、Mark Silberstein 和 Eran Tromer。通过寄存器缓存，在 GPU 上实现二进制域中的快速乘法。载于 2016 年国际超级计算会议论文集，第 1–12 页，2016 年。[33] Guyue Huang、Guohao Dai、Yu Wang 和 Huazhong Yang。GE-SpMM：面向图神经网络的 GPU 通用稀疏矩阵乘法。载于高性能计算、网络、存储与分析国际会议论文集，SC '20。IEEE 出版社，2020 年。[34] Md Rahman、Majedul Haque Sujon、Ariful Azad 等。FusedMM：面向图嵌入和图神经网络的统一 SDDMM-SpMM 内核。载于 IEEE IPDPS 第 35 届会议论文集，2021 年。[35] 技术报告——BitGNN：释放二值图神经网络在 GPU 上的性能潜力：https://tinyurl.com/yuf87cax，2023 年。[36] NVIDIA。CUDA 编程指南，2022 年。[37] Yue Zhao、Jiajia Li、Chunhua Liao 和 Xipeng Shen。弥合深度学习与稀疏矩阵格式选择之间的差距。载于第 23 届 ACM SIGPLAN 并行编程原理与实践研讨会论文集，2018 年。[38] Zhilin Yang、William Cohen 和 Ruslan Salakhudinov。重新审视利用图嵌入进行半监督学习。载于国际机器学习会议，第 40–48 页。PMLR，2016 年。[39] Weihua Hu 、 Matthias Fey 、 Marinka Zitnik、Yuxiao Dong、Hongyu Ren、Bowen Liu、Michele Catasta 和 Jure Leskovec。开放图基准：用于图上机器学习的数据集。神经信息处理系统进展，33:22118–22133，2020 年。[40] Oleksandr Shchur、Maximilian Mumme、Aleksandar Bojchevski 和 Stephan Günnemann。图神经网络评估中的陷阱。arXiv 预印本 arXiv:1811.05868，2018 年。[41] Marinka Zitnik 和 Jure Leskovec。通过多层组织网络预测多细胞功能。《生物信息学》，33(14):i190–i198，2017 年。[42] R. J. Fowlkes 和 D. G. Mallows。一种比较两个层次聚类的方法。《美国统计学会杂志》，78(383):553–569，1983 年。[43] Gang Chen、Haitao Meng、Yucheng Liang 和 Kai Huang。利用二值神经网络实现 GPU 加速的实时立体估计。IEEE 并行与分布式系统汇刊，31(12):2896–2907，2020 年。[44] Chun-Hsian Huang。面向农业应用的基于 FPGA、采用二值化神经网络的软硬件设计：案例研究。IEEE Access，9:26523–26531，2021 年。[45] Nael Fasfous、Manoj-Rohit Vemparala、Alexander Frickenstein、Lukas Frickenstein、Mohamed Badawy 和 Walter Stechele。BinaryCop：基于二值神经网络的边缘设备 COVID-19 口罩佩戴及位置预测器。载于 2021 年 IEEE 国际并行与分布式处理研讨会工作坊（IPDPSW），第 108–115 页。IEEE，2021 年。[46] Chao Ma、Yulan Guo、Yinjie Lei 和 Wei An。用于三维物体识别的二值体积卷积神经网络。IEEE 仪器与测量汇刊，68(1):38–48，2018 年。[47] Yinglan Ma、Hongyu Xiong、Zhe Hu 和 Lizhuang Ma。使用二值化神经网络实现高效超分辨率。载于 IEEE/CVF计算机视觉与模式识别会议研讨会论文集，第 0–0 页，2019 年。[48] Tong Geng、Tianqi Wang、Chunshu Wu、Chen Yang、Shuaiwen Leon Song、Ang Li 和 Martin Herbordt。LP-BNN：通过层并行实现超低延迟二值神经网络推理。载于 2019 年 IEEE 第 30 届应用专用技术国际会议。

系统、架构与处理器（*ASAP*），第 2160 卷，第 9–16 页。IEEE，2019。[49] Tong Geng、Ang Li、Tianqi Wang、Chunshu Wu、Yanfei Li、Runbin Shi、Wei Wu 和 Martin Herbordt。O3bnn-r：一种用于高性能和正则化二值神经网络推理的乱序架构。*IEEE 并行与分布式系统汇刊*，32(1):199–213，2020。[50] Angus Galloway、Graham W Taylor 和 Medhat Moussa。攻击二值化神经网络。*arXiv 预印本 arXiv:1711.00449*，2017。[51] Oliver Wieder、Stefan Kohlbacher、Mélaine Kuenemann、Arthur Garon、Pierre Ducrot、Thomas Seidel 和 Thierry Langer。图神经网络分子性质预测的简要综述。*Drug Discovery Today: Technologies*，37:1–12，2020。[52] Xiaoyang Wang、Yao Ma、Yiqi Wang、Wei Jin、Xin Wang、Jiliang Tang、Caiyan Jia 和 Jian Yu。通过时空图神经网络进行交通流量预测。载于*The Web Conference 2020* 论文集，第 1082–1092 页，2020。[53] Christian Nauck、Michael Lindner、Konstantin Schürholt、Haoming Zhang、Paul Schultz、Jürgen Kurths、Ingrid Isenhardt 和 Frank Hellmann。使用图神经网络预测电网的吸引域稳定性。*New Journal of Physics*，24(4):043041，2022。[54] Wenbing Huang、Tong Zhang、Yu Rong 和 Junzhou Huang。面向快速图表示学习的自适应采样。载于 S. Bengio、H. Wallach、H. Larochelle、K. Grauman、N. Cesa-Bianchi 和 R. Garnett 编，神经信息处理系统进展，第 31 卷。Curran Associates, Inc.，2018。[55] Petar Veličković、Guillem Cucurull、Arantxa Casanova、Adriana Romero、Pietro Liò 和 Yoshua Bengio。图注意力网络。发表于国际学习表征会议，2018年。[56] Ronald J Williams。用于连接主义强化学习的简单统计梯度跟踪算法。*Machine Learning*，8(3):229–256，1992年。[57] Fabrizio Pedersoli、George Tzanetakis 和 Andrea Tagliasacchi。Espresso：面向二值深度神经网络的高效前向传播。发表于国际学习表征会议，2018年。[58] Mir Khan、Heikki Huttunen 和 Jani Boutellier。用于 GPU 高效推理的二值化卷积神经网络。发表于*2018*年第*26*届欧洲信号处理会议（*EUSIPCO*），第682–686页。IEEE，2018年。[59] Haitao Meng、Chonghao Zhong、Jianfeng Gu 和 Gang Chen。用于实时高精度密集深度感知的 GPU加速的深度立体视觉与激光雷达融合。发表于*2021*年欧洲设计、自动化与测试会议暨展览（*DATE*），第523–528页。IEEE，2021年。[60] Chang Ye、Yuchen Li、Bingsheng He、Zhao Li 和 Jianling Sun。面向实时欺诈检测的 *GPU*加速图标签传播，第2348–2356页。计算机协会，纽约，美国，2021年。[61] Boyuan Feng、Yuke Wang、Tong Geng、Ang Li 和 Yufei Ding。APNN-TC：在 Ampere GPU张量核心上加速任意精度神经网络。发表于高性能计算、网络、存储与分析国际会议论文集，第1–13页，2021年。