PiPAD：在图形处理器上进行流水线并行动态 GNN 训练

王春阳*北京航空航天大学 中国北京
w
angchunyang@buaa.edu.cn

孙德森*北京航空航天大学 中国北京
sy2006344
@buaa.edu.cn

白月斌 北京航空航天大学 中国北京
byb@buaa.edu.cn

arXiv:2301.00391v2 [cs.LG] 5 Jan 2023

摘要

[1] 动态图神经网络（DGNN）已广泛应用于链接预测和疫情预测等各种现实应用，以从动态图中捕捉静态结构信息和时间特征。动态图神经网络结合了依赖时间和独立于时间的组件，具有巨大的并行计算和数据复用潜力，但在经典的逐图训练模式下，存在严重的内存访问低效和数据传输开销问题。为应对这些挑战，我们提出 PiPAD，这是一种用于优化图形处理器端到端性能的**流**水线**并**行**动**态图神经网络训练框架。PiPAD 从算法和运行时两个层面，对整体训练范式进行全面重构，涵盖数据组织和计算方式。PiPAD 能够并行处理多个图快照，从而消除不必要的数据传输并缓解内存访问低效，提升整体性能。在多个数据集上的评估表明，与最先进的动态图神经网络框架相比，PiPAD 在三种代表性模型上实现了 $1.22\times$ –$9.57\times$ 加速比。

1 引言

图神经网络（GNN）已被广泛用于处理图结构数据，并提取其中的潜在依赖关系 [1] ，应用于各种图相关任务，包括节点分类 [10], 、推荐 [25]和链接预测 [44]。在众多 GNN 变体中，例如图卷积网络（GCN） [20], ，一种常见且普遍采用的方法是结合图（聚合）操作和神经网络（更新）操作。聚合函数从每个节点的邻居收集并聚合特征向量，而更新阶段则利用全连接（FC）层等神经网络操作，使用聚合后的信息更新每个顶点。由于稀疏邻接矩阵的读取会产生不规则的内存访问和稀疏计算，类似稀疏矩阵-稠密矩阵乘法（SpMM）的聚合操作 [15] 通常被视为 GNN 的主要瓶颈，并引起了大量研究关注 [9, 15, 16, 48]。

在许多现实世界的场景中，例如金融交易[5],社交媒体 [22] 和分子生物学 [8],，图的拓扑结构以及节点特征可能会随时间动态演化。根据分区原则和粒度，动态图表示可分为两类 [5, 19, 51]：连续时间动态图（CTDGs）和离散时间动态图（DTDGs）。图神经网络研究社区提出了各种动态图神经网络（DGNNs），以捕捉移动图中的时间和结构信息。对于将动态图表示为以固定时间间隔采样的一系列快照的 DTDG，一种通用方法是在所有时间步上使用静态图神经网络（例如 GCN）对单个快照进行空间图学习，同时部署循环神经网络（RNNs）以获取不同快照之间的时间特征 [5, 12, 39]。此外，为了更好地捕捉时间依赖关系并提高准确率，滑动窗口机制也被广泛采用，该机制会同时向模型输入多个连续快照 [12, 32, 33, 39]。本文聚焦于**基于 DTDG 的**动态图神经网络（DGNNs），并为简便起见，将滑动窗口称为帧。

与传统图神经网络相比，动态图神经网络整合了处理大量快照的时序组件，这带来了**两个主要的性能问题**。首先，由于动态图神经网络训练需要沿时间线持续更新图快照，数据传输开销占据了整体训练时间的主导地位，并进一步加剧了因内存密集型图神经网络聚合操作而已经存在的图形处理器利用率不足问题。我们基于当前最先进的动态图神经网络框架 PyTorch Geometric Temporal（PyGT）开展的初步实验（§ 3.1） [38], 表明，通过 PCIe 进行的数据传输占据了总执行时间的近 39% ，而图形处理器利用率平均低于42%。其次，如 § 3.3 中的详细介绍所示，时间无关组件（图神经网络）与时间相关组件（循环神经网络）的组合产生了大量并行机会（例如，同时执行来自多个快照的独立操作），而相邻帧之间的快照重叠也提供了充足的数据复用机会。这些加速潜力亟待得到充分利用。

近年来，已有一些研究致力于在图形处理器上实现高效的动态图神经网络计算[5, 12, 22, 38, 51]，其中一些工作涉及上述问题。然而，相关解决方案要么主要关注大型图的扩展和分布式训练 [5] ，要么通过假设图拓扑（或

*两位作者对本工作贡献相同。[1]即将在 PPoPP'23 上发表。

节点特征）不会随时间变化 [12, 22], 来限制其输入场景，从而削弱了适用性。此外，这些方案都未能同时解决上述两个问题，以最大限度地优化端到端训练性能。

我们不再将这两个问题分开分析，而是从更全面的角度以协同的方式加以解决。这一思路源于两项关键发现。首先，真实世界的动态图通常变化缓慢，因此相邻快照之间存在大量拓扑重叠 [5]。其次，除类似 SpMM 的聚合所具有的不规则访问模式外，图神经网络还会因节点特征维度多样而遭遇其他类型的内存效率低下问题。对单个图执行聚合往往会导致带宽利用不足、线程利用率低或请求突发等问题（§3.2）。总而言之，经典的逐快照训练范式不可避免地导致大量冗余数据传输和低效的内存访问。因此，我们认为可以同时缓解 GPU 利用不足和并行度不足的问题。具体而言，开销巨大的聚合操作作用于邻接矩阵，而这些矩阵恰恰是冗余数据传输的主要来源。提取重叠的拓扑结构，并对多个快照执行一次与时间无关的聚合操作，不仅能够减少通信量，还有助于实现合并内存访问并解决带宽利用率低的问题。

为此，我们提出 PiPAD——一种流水线并行动态图神经网络训练框架，从算法和运行时两个层面优化单个 GPU 上的端到端性能。PiPAD 不对输入场景作任何假设，而是从数据组织、传输方法到计算方式，重构整个动态图神经网络训练过程，从而实现流水线化和并行执行。**首先**，利用动态图逐渐变化的特性，我们提出一种基于切片的图表示格式，以及一种感知重叠的多快照传输方法。该设计旨在缩短数据从 CPU 加载到 GPU 的时间，并促进高效并行计算。**其次**，我们设计了帧内并行机制和帧间复用机制，以充分释放动态图神经网络的加速潜力。前者通过线程感知切片合并实现维度感知并行图神经网络，从而加速聚合；同时，针对更新过程采用局部性优化的权重复用。我们的并行 GNN 支持同时处理多个快照，以提升计算性能并缓解内存访问低效问题。帧间复用则可最大限度地消除冗余传输和计算开销。**第三**，为优化整体训练性能，我们实现了一个流水线执行框架，用于协调 CPU 端操作、PCIe 数据传输和 GPU 计算。此外，PiPAD 还集成了一个运行时调节器，用于动态调整动态图神经网络训练的并行度和复用级别，以更好地适应各种数据集。

由于我们的目标是缩短端到端执行时间，因此我们在线执行额外的数据处理（图切片），并避免离线模型规模分析。具体而言，PiPAD 仅对并行图神经网络内核进行预先性能分析和分析，以指导在线调优。由于 GCN 是当今动态图神经网络中最常用的图神经网络 [39],，本文针对基于 GCN 的模型。但鉴于类似稀疏矩阵-稠密矩阵乘法的聚合是主流图神经网络[15, 16] （例如图注意力网络 [44]）的基础，我们的方法也可应用于各种类型的动态图神经网络。

总而言之，本文作出了三项贡献：

- 我们全面分析了三种典型动态图神经网络模型的执行与性能特征（§2.1 & § 3），揭示了动态图神经网络训练中的若干关键发现，并以此为设计提供依据。
- 我们提出 PiPAD，以优化单 GPU 动态图神经网络训练的端到端性能。通过结合重叠感知数据组织（§ 4.1）、帧内并行度（§ 4.2）、流水线执行机制（§ 4.3）以及采用动态调优技术的帧间复用（§4.4），PiPAD 全面实现了多快照方式下的流水线并行训练。
- 我们在各种数据集上进行的评估表明，与最先进的动态图神经网络框架相比，PiPAD 在三个代表性模型上实现了 $1.22\times - 9.57\times$ 加速比。

2 背景

2.1 动态图神经网络

动态时序动态图（DTDG）是由一组有序快照 $\{G^1, G^2 \cdots G^t\}$ 构成的，其中 $G^t = \{V^t, E^t\}$ 表示时间步 t 的快照，该快照包含顶点 V^t 和边 E^t。对于单个快照 G 和一个具有 K 层的图神经网络，令 h_v^{k-1} 表示顶点 v 在第 $k-1$ 层的特征向量，$N(v)$ 表示顶点 v 的所有一跳邻居。图神经网络在第 k 层上的计算模式可表示为公式 1。经过 K 层传播后，我们将得到每个顶点的最终特征。

$$h_v^k = \mathbf{Update}^k(\mathbf{Aggregation}^k(h_u^{k-1} | \forall u \in \mathcal{N}(v) \cup \{v\})) \quad (1)$$

不同的图神经网络在两个关键阶段所使用的具体函数上有所不同。在图卷积网络（GCN）中，聚合过程使用均值 函数处理收集到的特征，以获得每个顶点的聚合结果 a_v^k（即隐藏表示），而基于全连接层的更新则使用权重 w^k 和偏置参数 b^k 计算 h_v^k。

基于动态图的动态图神经网络遵循滑动窗口／帧处理模式，并可能包含多个层。如图 1所示，在每一层中，图神经网络组件从每个快照中学习结构信息，而循环神经网络模块则沿时间线捕捉时间特征。动态图神经网络可分为两类 [39]：堆叠式

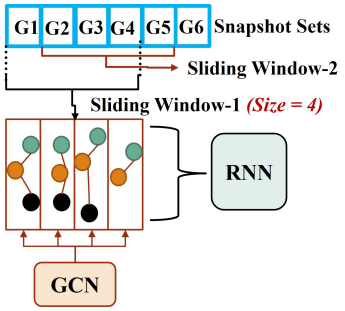
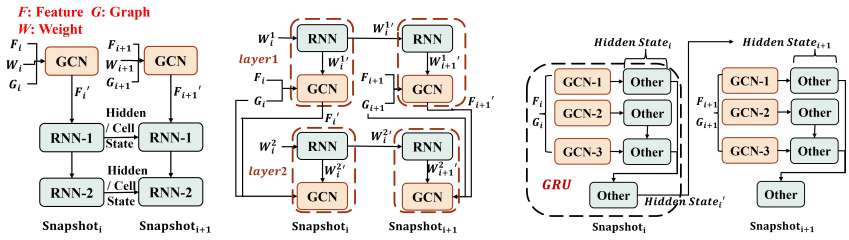


图1. 动态图神经网络执行流程。



(a) MPNN-LSTM (b) EvolveGCN (c) T-GCN

图2.动态图神经网络模型结构。→：数据传播。

和集成式。我们选择了三个有代表性的模型，用于本文其余部分的分析和评估。

MPNN-LSTM [32] (图 2(a)) 以简单方式堆叠一个两层图卷积网络和两个长短期记忆网络 [14] 模型。长短期记忆网络对图卷积网络生成的特征进行运算。多个快照之间唯一的数据依赖关系存在于长短期记忆网络的隐藏状态更新过程中。

EvolveGCN [33]（图 2(b)）在两个层中的每一层都包含一个 1 层图卷积网络和一个门控循环单元 [6]。第二层中的图卷积网络将第一层的输出作为输入特征。与 MPNN-LSTM 不同，EvolveGCN 将循环神经网络模块应用于图卷积网络的权重矩阵，使权重能够不断演化，从而产生跨快照依赖。该综述 [39]将 EvolveGCN 归类为集成型。

T-GCN [50]（图 2(c)）通过替换原有的通用矩阵乘法（通用矩阵乘法），将多个 1 层图卷积网络集成到门控循环单元中。隐藏状态沿时间线传播，形成与 MPNN-LSTM 类似的依赖关系。

2.2 图神经网络加速

静态图神经网络。过去几年里，静态图神经网络在图形处理器上的加速取得了大量研究成果，涵盖通用运行时框架 [7, 26, 47, 52]、类 SpMM 聚合优化 [9, 15, 16, 48] 以及分布式训练的扩展 [18, 23, 41, 43, 45, 46, 49]。

动态图神经网络。PyGT [38] 和 TGL [51] 是两个通用框架，旨在尽可能广泛地支持各种动态图神经网络模型。相比之下，DynaGraph[12] 和 CacheG [22] 专注于优化特定的一类模型。它们还发现了动态图神经网络中的并行与复用机会（§ 3.3），并分别利用时间步融合和中间结果缓存来提升性能。然而，DynaGraph 仅针对集成式动态图神经网络，并假设图拓扑不会随时间演变；CacheG 则要求节点特征保持不变。Cambricon-G [40] 和 ESDG [5] 都涉及动态图神经网络的数据传输开销问题（§ 3.1）。作为硬件加速器解决方案，Cambricon-G 设计了一种基于长方体的处理架构，支持细粒度数据传输，以避免不必要的快照拓扑更新。ESDG 采用类似的基于图差异的传输方法，减少大型图扩展和多 GPU 训练中的通信量。但这两项研究忽视了动态图神经网络固有的并行潜力，错失了实现进一步加速的机会。

图神经网络或传统深度神经网络的流水线/并行训练已广泛应用于分布式环境 [17, 27,28, 41, 45, 49] 以及并发多模型计算中 [3,42]。OOB [31] 和 Nimble [21] 是与我们相似的两项研究，它们支持在单个图形处理器上对单个深度神经网络进行并行训练。但前者的动机在于反向传播，而后者专注于利用 CUDA Graph [11] 和依赖预分析机制消除多流开销。它们的工作与我们的研究关注点正交。

总结。关于静态图神经网络类似 SpMM 的聚合优化的研究适用范围更广，也可应用于动态图神经网络领域。但有些研究 [15, 48]依赖繁琐的节点重排序（每个快照耗时最多可达数秒），可能会严重损害端到端性能。此外，这些方案和现有的动态图神经网络优化方法都无法同时解决这两个关键问题。

3 动机

本节首先通过采用我们的评估设置进行的初步实验（§5.1），揭示动态图神经网络训练的瓶颈及其底层执行效率低下的问题。随后，我们剖析数据复用和并行计算的潜力，以便同时解决这些问题。

3.1 动态图神经网络训练的性能瓶颈

数据传输开销。我们分析了使用 PyGT 进行动态图神经网络训练的整体性能特征。PyGT 构建于广受欢迎的图神经网络计算框架 PyTorch Geometric (PyG) [7]之上。与针对单个图的静态图神经网络相比，动态图神经网络处理快照序列，并且需要不断地将新的图数据从 CPU 加载到图形处理器。图3展示了与图形处理器相关的训练时间细分（左轴）和 SM 利用率（右轴）。如图所示，繁重的数据传输占据了总执行时间

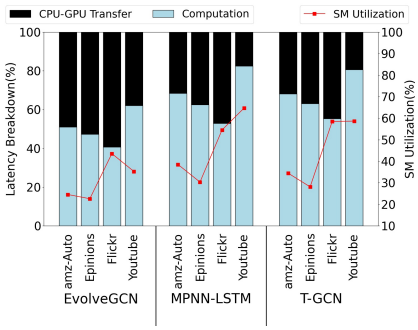


图3。动态图神经网络训练的延迟分解和SM利用率。

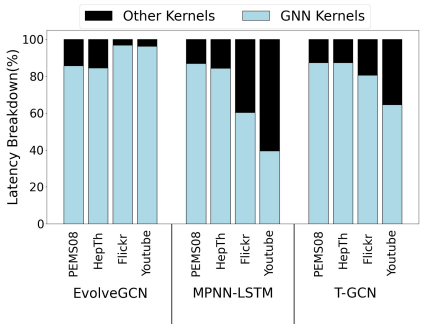


图4。动态图神经网络训练中的图形处理器计算时间分解。

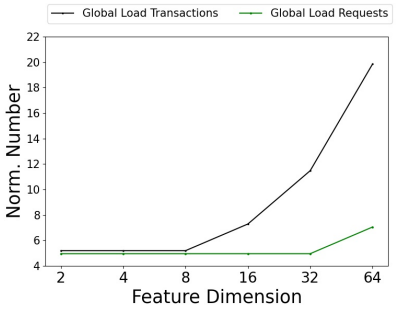


图5。全局内存请求数和事务数。

，平均占比为 38。7% 。随后，这导致图形处理器利用率严重不足：通过 PyTorch Profiler 测得的流式多处理器 (SM) 利用率 [35]平均低于 41。2% 。利用 CUDA 流实现异步数据传输，是缓解这一问题的一种直接且基础的方法。

拓扑重叠。现实动态图中相邻快照之间的拓扑变化率通常有限（在我们的数据集中，平均接近10%）。ESDG [5] 利用这一特点，采用图差异传输方法，仅更新拓扑中发生变化的部分，同时仍遵循一次训练一个快照的方式。

3.2 图神经网络中的内存访问低效

尽管它仍是动态图神经网络模型中的主要计算负担（图 4），图神经网络仍面临内存访问效率低下的问题。以往的图神经网络研究揭示了由输入图的稀疏性导致的聚合访问不规则性 [9, 15, 16, 48]。这些研究重构了稀疏邻接矩阵的访问模式，并据此实现局部性感知计算，以提高并行度。然而，稠密矩阵（输入或聚合后的特征向量）的维度也会导致其他类型的低效，而现有方案忽略了这一点。在一种常见的稀疏矩阵-稠密矩阵乘法实现中，每个外层迭代中，一个线程束负责处理稀疏矩阵中的单个元素，以及稠密矩阵中的整行。为利用局部性并减少昂贵的片外访问，GPU核心从全局内存加载特征，并将当前迭代的中间结果存储在共享内存中，以供下一次迭代复用。需要注意的是，主流图形处理器（例如英伟达）通常采用至少 32 字节的全局内存访问粒度。此外，按照单指令多线程（SIMT）模式，一个由 32 个线程组成的线程束通过一次请求最多可获取 128（32×4）字节。考虑到特征矩阵每行的字节长度等于其特征维度（F）的 4 倍，存在两类内存访问低效：

- **带宽利用不足**：当F小于 32/4 时，有效数据的大小小于单次事务的最小访问粒度。
- **请求突发**，其中 F大于128/4，这意味着线程束访问一行时，需要向共享内存和全局内存发出多个请求才能完成。

我们进一步通过使用 GNNAdvisor 运行 GCN 的实验来验证上述见解 [48], 。GNNAdvisor 是一项先进的 GNN 聚合优化工作。如图5所示，当特征长度小于 8 时，全局内存请求数（$\#R$）与事务数（$\#T$）几乎相同，且都几乎没有变化。随后，当维度超过 8 时，$\#T$开始增加，而当维度大于 32 时，$\#R$开始上升。考虑到现实生活中基于图的应用所使用的特征维度各不相同，这些问题显然亟待解决。此外，处理特征矩阵时若 F 小于 32，仍为每行分配一个线程束，这意味着执行过程中线程束内许多线程会处于空闲状态。线程利用率低的问题可通过线程束_执行_效率指标（§ 5.3）体现出来。

3.3 数据复用与并行度的机会

数据复用。帧机制的前向步长（图1）通常设为 1，以充分捕获所有快照之间的时间交互。因此，连续帧之间存在大量重叠区域，这意味着可以避免一些相关的冗余数据传输和计算。具体而言，第一层 GCN 中的聚合操作基于图拓扑的邻接矩阵和原始节点特征，与沿时间线进行的参数更新无关。我们可以缓存相关的聚合结果（a_v^k，见 § 2.1），并在下一帧或训练轮次中重复使用。

并行度。如图2所示，三种不同动态图神经网络都存在跨快照依赖，且这种依赖均位于时间相关的循环神经网络组件中。这意味着，可以并行执行不同快照上的相同图神经网络操作，从而提升性能。通过精心设计和优化，多快照处理方式有望一举实现两

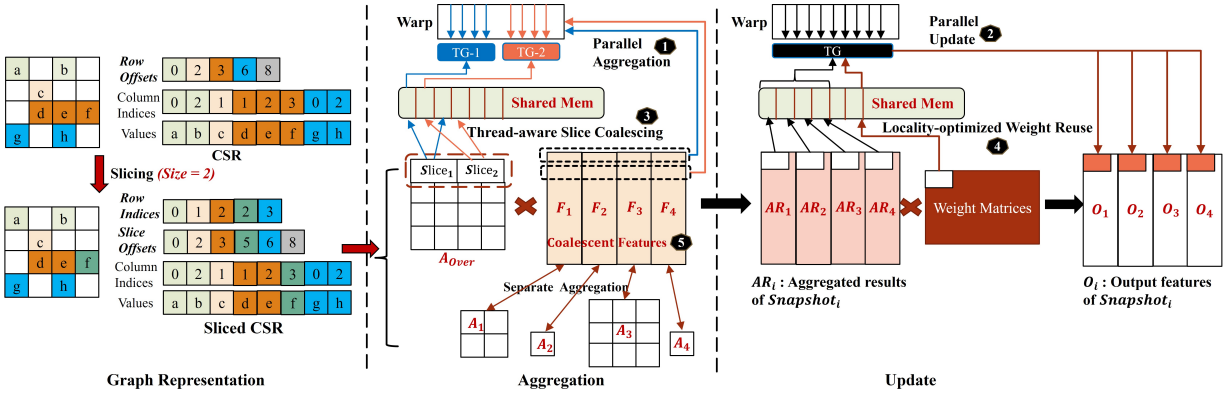


图 6。 切片图表示与并行GNN计算示例。 F_i： $Snapshot_i$的输入特征。A_i： $Snapshot_i$独占部分的邻接矩阵。 A_{Over}：重叠部分的邻接矩阵。TG：线程组。

全其美。具体而言，我们首先协同考虑拓扑重叠，并设计一种新型图格式，以便高效提取重叠区域，消除冗余传输。然后，我们**设计一种“并行”图神经网络计算模式，可同时处理多个图。其基本思路是利用一个快照组中所有快照的特征矩阵，直接在该组的重叠拓扑上执行聚合**。这样，我们就能对多个特征实现合并内存访问，并缓解带宽利用不足的问题。此外，**在维度较大的情况下采用灵活的访问粒度，可以避免请求突发**。

4 PiPAD

我们对 PiPAD 进行了整体设计与实现，以提升端到端动态图神经网络训练的性能。我们的优化分为三个方面：通过重叠感知的数据组织和帧间复用减少数据传输量；利用帧内并行加速图神经网络计算；通过流水线最大限度地增加不同操作之间的重叠。本节将详细介绍这些机制。

4.1 重叠感知的数据组织

通过提取相邻快照之间的重叠拓扑，并为其构建新的独立邻接矩阵，我们可以避免不必要的数据传输，进一步实现并行计算的构想。由于帧机制，一个快照可能会与其他独立快照合并，在不同帧中进行处理。因此，在整个训练轮次启动期间，都会持续进行重叠分析与提取。压缩稀疏行（CSR）是广泛使用的稀疏邻接矩阵存储格式，它以较粗的行粒度和紧密有序的布局来管理图（图 6）。这些特性直接限制了重叠提取的灵活性，导致效率低下。此外，真实图中的稀疏性很容易造成稀疏矩阵乘法计算中的负载不均衡

因为在 CSR 表示方式下，通常会分配一个线程束来处理邻接矩阵的一行或几行。

基于切片的图表示。 为了支持更细粒度的存储并应对上述挑战，有两种选择：二维分块和一维切片。前者（例如块压缩稀疏行 [4]）将原始矩阵划分为多个较小的矩阵，而后者则对每一行进行划分，将整个矩阵转换为多个切片。由于每个分块子矩阵中的非零元素（nnz）分布离散，导致稀疏性更加不均匀，因此，切片方式不仅空间开销更小，还能通过压缩实现更好的负载均衡。因此，我们设计了图 6所示的切片压缩稀疏行格式。由于单个切片位于同一行内，我们将压缩稀疏行格式中的原始行偏移量数组改为行索引（RI），并新增一个数组切片偏移量（SO）。RI 表示每个切片的行索引，SO 则存储列索引字段中每个切片首个元素的偏移量。我们以切片粒度管理整个图，每个切片存储一定数量的非零元素，且有一个上限（图 6中为 2）。这有助于快速构建切片粒度的重叠，并提升计算中的负载均衡。

重叠感知的数据传输。 后文将介绍，PiPAD 通过将每一帧划分为多个分区 来实现并行训练，每个分区包含连续的快照。因此，我们将数据传输调整为以分区为粒度的模式。我们将拓扑数据（邻接矩阵）重新组织为所有快照共用的一个重叠部分，以及每个快照各自的若干独占部分，以减少传输量。随后，我们利用独立的 GPU 流和固定内存，按需实现异步数据传输。

空间开销。 主要开销来自修改后的行索引和新增的切片偏移量。设 $\#nnz$ 表示矩阵中的非零元素数量。我们的切片 CSR 需要 $(2\times\#nnz+2\times\#Slice+1)$ 空间，而 CSR 需要 $(2\times\#nnz+\#vertices+1)$。PyG 采用的另一种常用图表示方式——坐标格式（COO），使用三个数组分别存储所有非零元素的值、行索引和列索引，其 $(3\times\#nnz)$

空间用量。我们设置每个切片最多容纳 32 个非零元素，因此空间开销通常介于 CSR 和 COO 之间。

4.2 帧内并行

我们基于维度感知并行图神经网络实现帧内并行，使其能够同时对多个快照执行聚合和更新操作（❶和❷ 见图 6）。此外，我们设计了三项关键优化：针对大维度场景的基于向量内存指令的内存访问、针对小维度场景中线程利用率低的问题而提出的线程感知切片合并(❸)，以及用于实现高效并行更新的局部性优化的权重复用(❹)。

维度感知并行图神经网络。如 §3.3所述，通过多快照计算方式和灵活的访问粒度，可以缓解两类内存访问低效问题。首先，将不同快照的特征矩阵合并后执行一次聚合，能够更高效地利用带宽，尤其适用于节点特征维度较小的图。遵循重叠感知数据组织，我们将原始聚合拆分为两部分：对重叠区域的相邻矩阵与某个快照组的全部特征进行并行聚合（称为合并后的特征：❺ 见图 6），以及分别使用各个快照的特征对独有拓扑部分进行非并行聚合。其次，利用 [24] 支持单次请求加载 32/64/128 个浮点数的向量内存指令，是解决请求突发问题的一种可行方案。基于这些具有不同数据获取宽度的向量指令，在处理特征维度较大的图（大于 32）时，我们能够以更高效、粒度更大的方式加载数据。稍后我们将结合切片合并介绍聚合的实现。

线程感知切片合并。即使将多个特征矩阵合并为一个合并特征矩阵，总行长度（维度）仍可能小于 32，线程利用率低的问题依然存在。在这种情况下，为了进一步增加每个线程束中的活跃线程数，我们将邻接矩阵中相邻的切片合并为一组，并将其设为每个线程束的基本处理单元。相应地，同一线程束中的线程均匀划分为若干线程组（TGs），并为每个线程组分配一个切片，使其独占处理该切片。随后，我们在共享内存中以交错模式实现切片组数据布局（❸ 见图 6），从线程束的视角来看，单次数据加载的访问地址因此是连续的。每次计算迭代中，每个线程组都会从对应切片中处理一个元素，并从合并特征矩阵中处理一行。切片组的最大大小（$coalesce_num$），即每个线程束中的线程组数量，设为 4，以确保每个线程组的数据访问粒度不超过单个内存事务的长度（32 字节）。算法1 详细介绍了我们针对小维度情况的并行聚合过程，其输入包括

算法 1： 并行聚合与切片合并

输入： 特征，切片_大小，合并_数量
输出： 聚合_结果

```
1  lane_id =获取线程束内索引()
2  /* 计算分配给当前线程的特定切片的索引，用于
       加载和计算*/
3  load_index = lane_id % coalesce_num_
4  comp_index = lane_id / features.dim
5  /* 计算分配给
       当前线程的合并特征维度，用于加载和计算 */
6  comp_dim = lane_id % features.dim
7  /* 检查当前线程是否分配了工作*/
8  如果 comp_index < coalesce_num那么
9      /*计算线程在切片中的具体工作负载大小*/
10     load_size, comp_size =compWork(load_index, comp_index)
11     /*加载切片中的目标元素*/
12     对于 nid ← lane_id，执行至 (slice_size * coalesce_num)
13         /*计算加载时切片中的元素偏移量*/
14         load_offset = nid / coalesce_num
15         如果 load_offset > load_size 则
16             跳出
17         end
18         将切片加载到共享内存(load_offset)
19         nid += coalesce_num * features.dim
20     end
21     /*加载特征中的目标维度，然后进行计算*/
22     对于 idx ← 0 到 comp_size 执行
23         /*计算切片中的元素偏移量以进行计算*/
24         comp_offset = idx * coalesce_num + comp_index
25         Slice =从共享内存加载切片到寄存器(comp_offset)
26         Feature =从全局内存加载特征到寄存器(comp_dim)
27         乘加(Slice, Feature, partial_result)
28         idx ++
29     end
30     原子加(aggregation_result, partial_result)
31 结束
```

合并特征矩阵、每个切片拥有的最大非零元素数（32）以及 $coalesce_num$。在实现中，首先为每个线程分配任务，使其以固定步长从全局内存加载一个特定切片中的元素到共享内存，并且切片索引与线程索引直接对应（第 3 行和第 9–20 行）。这一设计基于前文所述的交错数据布局，以确保每个线程束能够连续访问内存。随后，线程获取分配给它的特征中特定维度所对应的数据（第 6 行和第 26 行），按照相同的步长模式从共享内存中加载目标切片的元素，最后执行计算（第 4 行和第 21–30 行）。

局部性优化的权重复用。为配合并行聚合，我们采用权重复用原则，以实现局部性更好的并行更新。我们不再对一个快照直接执行通用矩阵乘法，然后再处理下一个快照，而是让一个权重块留在共享内存中，依次与所有快照的特征进行计算（❹），然后在同一轮迭代中切换到下一个权重块。这种经过局部性优化的计算范式能够最大限度地复用权重。请注意，由于 Evolve GCN 会沿时间线更新权重，因此无法应用此技术。

对于动态图神经网络中的其他内核，我们按照计算和数据依赖关系的顺序依次执行。此外，

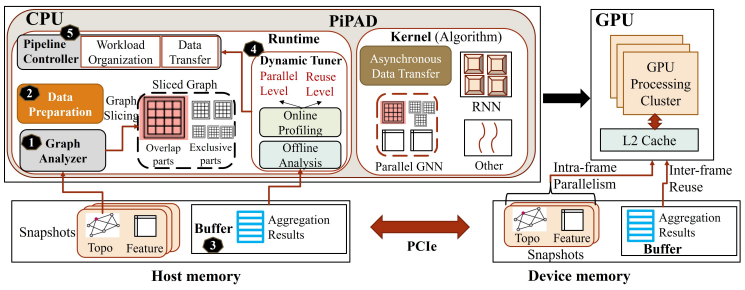


图 7. PiPAD 的整体架构。

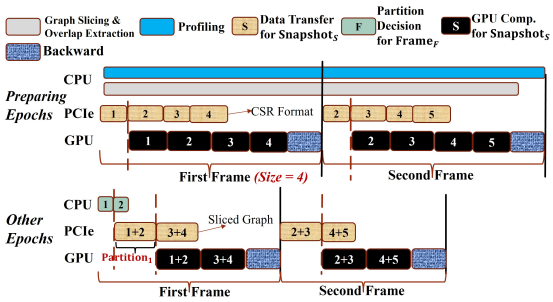


图 8. 流水线执行示例。

我们参考 OOB [31] 并利用 CUDA Graph API [11]将这些内核一起启动，从而降低启动开销。

4.3 流水线执行框架

如图 7所示，PiPAD 由两部分组成：用于高效进行多快照计算的**算法级**优化，以及用于协调整体训练过程的若干**运行时级**组件。

具体而言，在算法层面，我们设计了切片 CSR，以便于提取图重叠部分；并重构图神经网络，以支持在一个帧内对多个快照进行并行计算。在运行时层面，为适配新的图格式，我们首先实现了一个低开销的在线图分析器（❶ 见图 7），将所有快照的格式从原始 CSR 高效转换为我们的格式。每个分区的邻接矩阵数据包含公共重叠部分，以及各个快照独有的部分。我们使用数据准备模块❷ 提取相邻快照之间的重叠部分，并为分区级计算做好准备。上述两个组件只需在最初几个训练轮次（即准备阶段）运行一次。在后续训练轮次中，PiPAD 会针对每个帧，以分区粒度执行并行训练。为支持不同帧之间的数据复用，我们维护*CPU* 和 *GPU* 侧缓冲区，❸ 用于缓存聚合结果。在运行时的上层，我们实现了一个动态调优器❹，根据并行图神经网络内核的离线分析结果，以及准备阶段的在线性能分析，在线调整并行度级别（一个帧内每个分区包含的快照数）和复用级别（GPU侧缓存的聚合结果大小）。需要注意的是，初步离线实验（§4.4）仅涉及图神经网络部分，用于进行敏感性分析。最后，我们的流水线控制器❺负责调控整体训练过程。

图8展示了 PiPAD 的流水线执行示例，其中每个分区包含两个快照，帧大小为 4。在**准备**阶段，训练遵循传统的单快照方式，并进行异步数据传输。在主机上，我们收集必要的统计数据，包括每个快照/帧的输入数据大小、执行时间和最大内存消耗。

性能分析在线执行，以避免早期离线模型规模测试，并摊销提升端到端性能所需的开销。这些数据将用于指导动态调优器选择优化后的并行选项，而不会触发内存不足（OOM）异常（§4.4）。此外，由于提取图的重叠区域并准备按分区划分的邻接矩阵也需要时间，我们在开始时一次性执行图切片及后续提取。每个分区的快照数配置是一个有限集合（例如 2、4 和 8），因此提取开销可控。同时，我们还记录每个帧内快照之间的图演变速率，以供后续在线调优决策使用。随后，在**接下来的**训练轮次中，训练以分区方式进行，从而降低数据加载开销并提高计算并行度。与规范模式相比，我们首先确定当前帧的并行度（§4.4），然后传输按分区划分的训练数据。由于不同训练轮次之间的性能相近，我们只执行一次此过程，并在之后的每个训练轮次中沿用生成的配置。

开销分析。PiPAD 会产生一定的时间和空间开销。首先，基于切片的图格式需要额外的空间（§ 4.1），并且在准备阶段需要执行辅助图切片过程。我们还需要提前提取重叠部分，为每个分区构建基于重叠的邻接矩阵。值得注意的是，我们的切片策略以细粒度管理图，并能加快提取过程。根据我们在评估设置下的实际测量，这些预处理可以在前两个训练轮次内完成。其次，只有在分区决策完成后，才能开始传输每个分区的数据。由于这项一次性操作在 CPU 上异步执行，并与前一分区/帧的其他操作重叠（图 8），因此其开销也可以摊销。

4.4 帧间复用与动态调优

本节介绍我们针对不同帧间的数据复用设计，以及对两个关键参数进行的动态调优。

帧间复用。如3.3节所述，第一层图卷积网络中的聚合操作不受沿时间线更新参数的影响。因此，一种直接的方法是将准备阶段中得到的所有相关聚合结果存储到CPU内存中，并在后续训练轮次中复用。这样可以避免反复传输那些重叠快照的邻接矩阵（对于仅包含单层图卷积网络的模型），并消除冗余的聚合计算。但这些聚合结果仍需传输到图形处理器，以供下一帧使用。

为进一步提升复用级别，我们根据下一帧中的使用顺序，在图形处理器端维护一个缓冲区，用于缓存部分聚合结果。缓冲区的最大容量受动态图神经网络训练过程的内存消耗和GPU内存容量限制。根据准备阶段的训练轮次中通过性能分析获得的逐帧内存使用统计信息，我们会根据需求为每一帧动态分配不同大小的缓冲区。请注意，由于*malloc/free*函数的执行耗时较长，只有在缓冲区原有大小不足以容纳新的中间结果时，我们才会重新分配缓冲区。

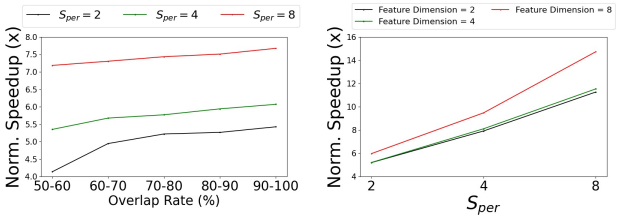
动态调优。为降低在线决策的复杂度，我们将单帧中的快照均匀分配给各个分区。随后，动态调优器需要为每一帧确定两个关键参数：每个分区的快照数（S_{per}），以及为下一帧复用而缓存在GPU中的聚合结果大小。后者根据上述内存使用情况确定。关于 S_{per}，有三个影响因素：

- **内存消耗**：与单快照模式相比，多快照计算会在某个阶段增加 GPU 内存使用量，因为处理每个分区都需要 GPU 保存所有相关快照的数据。我们应避免发生 OOM异常。
- **并行GNN的计算加速比**：我们的帧内并行度所带来的加速效果可能因输入图和 S_{per} 的取值而异。
- **计算与数据传输之间的重叠程度**：多快照处理不仅能提升计算阶段的性能，还会增大单次传输的数据量，进而可能导致流水线停滞，反而降低 GPU 利用率。

我们通过以下流程确定 S_{per} 的取值。

首先，由于我们采用了感知重叠的数据组织方法，且训练数据的大小有所减少， N 快照模式下的最大内存消耗不会超过单快照计算模式的 N 倍。利用我们的在线性能分析统计，可以推导出一个上限 U，确保我们的选择不会触发 OOM。

其次，计算加速比主要与三个因素相关： S_{per}、单个图的节点特征维度，以及所有快照之间的拓扑重叠率 (OR)



(a) 不同 S_{per} 设置相对于单快照执行的加速比随 OR 变化的情况。(b) 不同 S_{per} 设置的归一化加速比随特征维度变化的情况。

图 9. 并行GNN的离线分析结果。

分区内。同时运行更多快照通常意味着更高的并行度，但 OR 较低。由于帧内并行的主要优化在于利用重叠区域实现高效聚合，因此低 OR 带来的影响不容忽视。因此，我们对并行GNN进行离线分析，以指导在线决策。图 9(a) 和9(b) 分别展示了在不同 OR 和特征维度条件下，各种 S_{per} 设置在我们的数据集上取得的平均加速比。我们通过随机选择满足目标重叠要求的快照组来构建 OR 配置。结果清楚地表明，在 OR 或特征维度设置相同的情况下，更大的 S_{per} 更受青睐。更重要的是，按 OR 区间划分的粗略加速比分布有助于我们估算运行时不同S_{per} 设置的加速效果。我们利用离线分析结果以及准备阶段在线获取的实际逐帧拓扑变化率统计数据，来推算所有可用 S_{per}选项的执行延迟。

第三，我们先根据分析得到的数据大小信息，分析每种方案的数据传输延迟。然后，剔除会导致流水线停滞的方案，并从剩余方案中选择计算加速比最高的方案。对于每一帧，我们都会指定并行度和复用选项，随后由流水线控制器统筹整体训练。

4.5 讨论

我们基于 PyTorch [34] 和 C++、CUDA C 与 Python 实现了 PiPAD。但我们的方法具有通用性，不依赖于特定的深度学习或图神经网络框架。

局限性。多快照并行训练不可避免地会增加基于阶段的内存消耗，并导致大型图固有的扩展性问题。由于我们的切片 CSR 便于进一步划分图，因此可以通过扩展 PiPAD 以支持多图形处理器训练来解决这一局限性。我们将其留待未来工作。

表 1。用于评估的图数据集。展示了所有快照中的顶点数（#N）、边数（#E）、特征维度（D）、边寿命为 [5] 时图平滑后的边数（#E-S），以及快照数（#S）。

数据集	#N	#E	D	#S	#E-S
社交网络 [36, 37]					
Flickr	2.3 M	33.1 M	2	132	480 M
YouTube	3.2 M	602 K	2	198	11 M
电子商务 [36, 37]					
amz-Automotive	1.1 M	1.3 M	2	524	55 M
Epinions	727 K	13.6 M	2	99	78 M
引文网络 [36, 37]					
HepTh	22 K	2.6 M	16	214	18 M
交通网络 [2]					
PEMS08	170	7202	16	90	7202
疾病传播 [32]					
Covid19-England	130	82 K	16	61	108 K

5 评估

5.1 实验设置

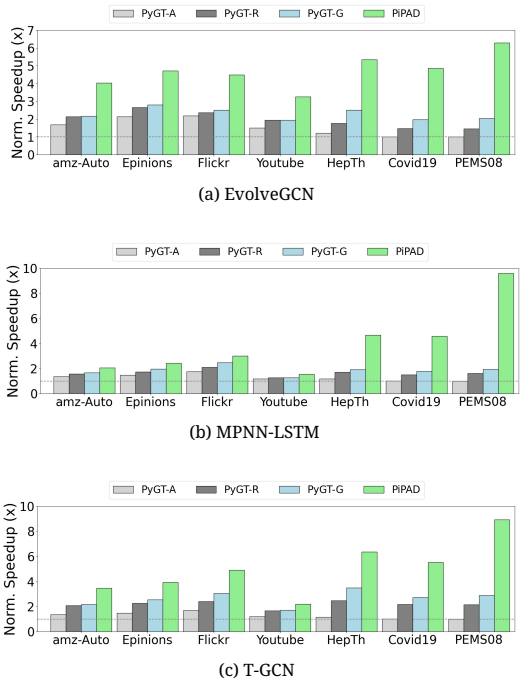
模型与数据集。我们选择了三个具有代表性的动态图神经网络模型（§2.1）：MPNN-LSTM [32], T-GCN [50] 和 EvolveGCN[33]。如表1所示，用于评估的数据集涵盖了多种真实世界应用，取自 Network Repository [36, 37]（ESDG [5]、ASTGNN[13]和 MPNN-LSTM [32]使用的数据集）。考虑到单个 GPU 的内存容量限制，我们将小型和大型图数据集的输入特征维度分别设为 16 和 2，隐藏维度则分别设为 32 和 6。

基线。由于 DynaGraph [12] 和 CacheG [22] 限制了输入场景，而 ESDG [5] 面向分布式训练，因此我们将 PiPAD 与基线**PyGT**及其三个变体进行比较：(1) **PyGT-A**：PyGT 的增强版本，支持异步数据传输；(2) **PyGT-R**：将我们的帧间复用机制集成到 PyGT-A 中的方案；(3) **PyGT-G**：与 PyGT-R 相比，PyGT-G 将原始 PyG 版本的图卷积网络模块替换为 GE- SpMM [15],，这是一种先进的聚合优化工作，无需对 [48] 和 [16] 等图进行预先节点重排序。GE-SpMM 利用共享内存缓存稀疏矩阵中的行，从而提高基于压缩稀疏行的聚合计算的局部性。我们采用这种渐进式对比设计，同时评估整体性能和各项单独优化技术。

环境设置与指标。我们的硬件平台是一颗 24 核 Intel(R) Xeon(R) E5-2680 v3 CPU，配备 128GB 主机内存和一块拥有 16GB 高带宽内存的 NVIDIA Tesla V100 GPU。帧大小设为 16，所有实验均在搭载 CUDA 11.3 的 Ubuntu 18.04 上进行。PyGT 和 PyTorch 的版本分别为 0.53.0 和 1.10。在整体性能对比中，我们训练 200 轮，并测量包括图切片、数据加载和 GPU 计算在内的端到端训练时间，以及硬件利用率。我们还对并行GNN和切片 CSR 进行了详细分析。

5.2 整体性能比较

图10 展示了三种模型相较于 PyGT 的端到端训练时间加速比。总体而言，凭借涵盖数据组织、数据传输和计算方式的整体设计，PiPAD 的表现优于所有对比方法，并实现了1.54 × −9.57× 相较于基线 PyGT 的提升（4.71×、3.98× 和 5.18×，分别为 EvolveGCN、MPNN-LSTM 和 T-GCN 的平均提升幅度）。由于小规模数据集（HepTh、PEMS08 和 Covid19）的顶点较少，特征向量规模也较小，我们的拓扑感知数据传输缩减和计算优化能够发挥更大的作用。此外，由于节点数量庞大且总内存使用量较高，PiPAD 在大规模数据集的评估中只能启用 2 快照并行，这限制了我们的加速空间。因此，我们的加速比通常在小规模数据集上更高；而 PyGT-A 则呈现相反的特点，因为异步传输是其唯一的优化。



(a) EvolveGCN

(b) MPNN-LSTM

(c) T-GCN

图10. 相较于 PyGT 的训练加速比。

由于采用了三重优化，PyGT-G 在几乎所有场景中都取得了第二好的性能：通过异步数据传输降低传输开销，通过帧间复用消除冗余，并通过 GE-SpMM 加速开销较大的聚合。与 PyGT-G 相比，我们的主要优势在于帧内并行度，它可以

表2. 不同方法的图形处理器利用率（%）。AA：amz-Automotive；EP：Epinions；FL：Flickr；YT：YouTube；HT：HepTh；CE：Covid19-England；PE：PEMS08。小规模数据集上的利用率较低，是因为 CPU 端延迟相对较高。

方法	EvolveGCN							MPNN-LSTM							T-GCN						
	AA	EP	FL	YT	HT	CE	PE	AA	EP	FL	YT	HT	CE	PE	AA	EP	FL	YT	HT	CE	PE
PyGT	75.18	70.75	76.47	78.85	31.9	12.04	10.87	84.81	80.44	81.36	89.91	36.01	11.89	10.99	83.8	79.56	81.93	87.67	35.41	13.15	11.92
PyGT-A	93.45	92.47	99.08	94.94	35.67	12.16	10.95	96.42	96.31	**99.38**	97.94	39.14	11.92	11.0	94.44	93.45	99.08	95.92	37.43	13.19	11.98
PyGT-R	**94.78**	**94.45**	**99.08**	**96.03**	**45.81**	**13.6**	**11.52**	**97.3**	**96.91**	99.16	**98.65**	50.17	**13.8**	12.51	**95.39**	94.5	**99.13**	97.72	**50.96**	**15.36**	13.59
PyGT-G	78.86	91.33	88.16	91.98	34.86	8.81	8.44	91.05	94.28	93.5	94.27	42.57	8.97	8.9	89.5	93.55	92.25	93.46	40.64	10.47	11.23
PiPAD	87.75	91.8	92.28	94.12	36.25	11.75	10.91	96.31	95.13	93.96	98.55	**67.92**	12.6	**14.25**	94.39	**94.68**	92.33	**98.07**	50.61	12.18	**14.23**

不仅能处理多个快照以提高吞吐量，还能通过复用权重来促进更新函数的执行。PyGT-R 的性能与 PyGT-G 非常接近，有些情况下甚至更好。首先，采用帧间复用可直接跳过第一个 GCN 层中的聚合，因此，旨在加速聚合的 GE-SpMM 在同时执行多个 GCN、行为上却如同仅包含一个 GCN 的 T-GCN 中几乎毫无用处。其次，GE-SpMM 在其实现中要求同时使用压缩稀疏行和压缩稀疏列（CSC）格式存储图，以便进行反向传播。这会增加传输开销，并损害其在 YouTube 等大规模数据集上的性能。

具体的加速比在不同情况下差异很大。接下来，我们结合图 3 和图 4中的实验结果，从模型角度进行详细分析。

EvolveGCN。 EvolveGCN包含两层，每层各有一个图卷积网络；即使采用数据复用优化，仍需在第二层图卷积网络中执行聚合。这为我们的帧内并行提供了更大的加速空间。此外，EvolveGCN中图神经网络模块之外的其他内核仅占总计算时间的一小部分（图 4），这进一步放大了并行GNN的效果。因此，在这三个模型中，PiPAD相较于排名第二的PyGT-G实现了最高的平均加速比（2.13×）。

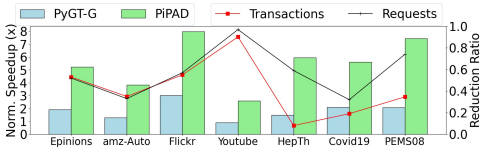
MPNN-LSTM。 与另外两个模型相比，在顶点数量庞大的较大数据集（例如图 3和 4中的YouTube）上，MPNN-LSTM的计算占比更高，且耗时更长的循环神经网络组件也更多。因此，在处理大规模数据集时，PiPAD在所有情况下的加速比最低（相较于PyGT-G，最低为 1.22×）。但在较小的数据集上，MPNN-LSTM中与循环神经网络相关的执行延迟要低得多（图 4）。此外，MPNN-LSTM采用两层图卷积网络，但不像EvolveGCN那样沿时间线更新权重。在这些小规模数据集上，我们的并行GNN对聚合和更新操作的加速效果都能充分发挥（相较于PyGT，最高为 9.57× ）。

T-GCN。由于多个 GCN 并行执行，借助帧间复用，T-GCN 中的所有聚合操作都被消除。因此，PyGT-R、PyGT-G 和 PiPAD 都获得了显著的加速比。但如上所述，PyGT-G 中 GE-SpMM 的效果也会随着帧间复用而被消除，而 PiPAD 仍可利用局部性优化的权重复用来加速更新阶段。

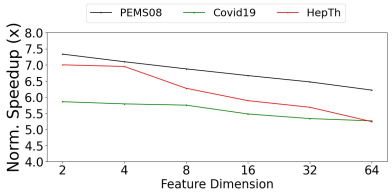
由于 PyTorch Profiler 不支持 CUDA Graph，我们使用 NVIDIA 系统管理接口 [30] 来收集图形处理器利用率数据，其中运行内存复制内核也会计入最终利用率（表 2）。因此，PyGT-R 和 PyGT-A 的表现反而更好，这是因为 PiPAD 和 PyGT-G 大幅缩短了计算时间（§ 5.3），放大了 CPU 端操作所占总时间的比例。不过，通过并行GNN减少已发起内核的数量，我们仍然优于 PyGT-G。

5.3 并行 GNN 分析

由于数据传输会显著影响端到端训练时间，尤其是在大规模数据集上，本节特别分析我们的算法级优化：帧内并行。我们统计了整个 DGNN训练过程 中 GNN（1 层）模块的执行时间和内核级全局内存访问情况。我们选择 PyGT 和 PyGT-G 进行比较。为全面揭示 GNN 的特性，我们禁用了帧间复用。



（a）图神经网络执行时间加速比（左轴）和内存访问性能（右轴）



（b）特征维度发生变化。

图11. 详细分析。

加速比。图 11(a)（左轴）展示了相对于 PyGT 的 GNN 执行时间加速比。与 GE-SpMM 相比

与采用低效的单快照方式且仅针对类似 SpMM 的聚合不同，我们的并行GNN 会同时处理多个快照，并优化聚合阶段和更新阶段。与 PyGT 和 PyGT-G 相比，我们分别实现了平均 5.6×和 3.1× 的性能提升。从数据集的角度来看，由于 PiPAD 和 GE-SpMM 都使用共享内存缓存聚合过程中来自相邻矩阵的元素，因此，与 PyGT-G 类似，我们在图结构局部性更好、图更稠密的数据集（如 Flickr、HepTh 和 Epinions）上取得了更高的加速比。

内存效率。与 GE-SpMM 仅利用共享内存优化邻接矩阵的访问不同，我们的并行执行能够实现对多个特征矩阵的合并内存访问。对于具有 2 维特征的大型数据集，并行GNN可以通过一次事务同时为两个图加载 16 字节特征，而 GE-SpMM 需要两次独立事务才能完成相同操作。对于具有 16 维特征的小型数据集，在 GE-SpMM 下，单次 64 字节数据加载会为一个图生成一个内存请求和两次事务。而借助向量内存指令，PiPAD 可以通过一次内存请求和八次事务，同时为四个图加载 256 字节数据。我们以更大的粒度访问内存，同时避免请求突发。因此，相较于 PyGT-G，我们将全局内存请求和事务总数平均分别降低了 57% 和 45% （图 11(a) 右轴）。在 YouTube 数据集上，性能提升尤为明显，这是由于该数据集具有极高的稀疏性。CSR 中大量的空行导致 GE-SpMM 产生大量冗余内存访问，而我们的切片 CSR 则通过更细粒度的管理方式避免了这一问题。

线程利用率。通过 NVIDIA 性能分析器测得的线程束_执行_效率指标，直接反映了每个线程束中平均活跃线程数的比例。为了更好地展示线程利用率低的问题，我们将所有七个数据集的输入维度和隐藏维度分别设为 2 和 6。评估结果表明，PyGT-G 中与图神经网络相关的内核平均线程利用率为 57.2%，而基于我们的线程感知切片合并，PiPAD 达到了 64.9%。

敏感性。我们进行了维度敏感性测试，以验证并行图神经网络的维度感知能力，结果如图 11(b)所示。受测试平台图形处理器内存容量限制，我们仅测试了三个小规模数据集。通过结合向量内存指令和线程感知切片合并，我们的并行图神经网络在各种特征维度设置下均能实现可观的加速比（至少 5.2×）。但受内存消耗因素影响，在小维度情况下，我们可以启用更高的并行度（更大的 S_{per}，见 § 4.4），以获得更高的加速比。

5.4 切片 CSR 分析

我们还使用原始 CSR 实现了 PiPAD 的一个变体，并开展对比实验来分析我们的切片 CSR。参照 [16],中的方法，我们绘制了

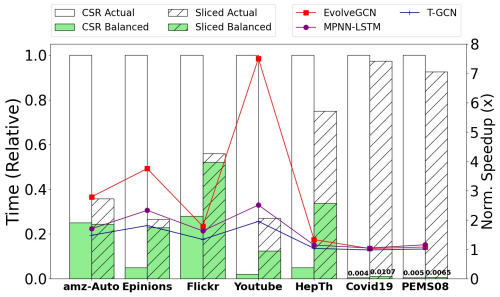


图12. 切片 CSR 的负载均衡分析与整体性能比较。

我们提升图神经网络内核负载均衡的效果，如图12 （左轴）所示。Balanced 柱（绿色部分）表示在负载完全均衡时的理想执行延迟，其计算方法是将所有线程块的总时间除以图形处理器可容纳的最大活跃线程块数 [16]。该理想值与实际执行时间（Actual 柱）之间的差距反映了负载不均衡程度，而我们的切片 CSR 在所有数据集上都降低了这种不均衡。对于小规模数据集，改进效果不那么显著，因为这些数据集更稠密，即使使用原始 CSR，也容易在计算过程中实现负载均衡。此外，从模型角度来看，整体训练时间的加速比（右轴）呈现出与负载均衡结果相似的趋势。具体而言，EvolveGCN 在 YouTube 上的显著改进，主要得益于 EvolveGCN 中图神经网络内核占比较大的执行时间（计算），以及由于 YouTube 极度稀疏，切片 CSR 比 CSR 占用更少空间（数据传输）。

6 结论

本文提出了 PiPAD，这是一种流水线并行动态图神经网络训练框架，旨在优化图形处理器上的端到端性能。通过高效的并行多快照处理和运行时级流水线编排，PiPAD 解决了动态图神经网络计算中数据传输过多、并行度未充分利用以及内存访问低效等问题。我们的评估表明，PiPAD 相较于最先进的动态图神经网络框架实现了 1.22×–9.57× 加速比。

参考文献

[1] Sergi Abadal、Akshay Jain、Robert Guirado、Jorge López-Alonso 和 Eduard Alarcón。2021年。图神经网络计算：从算法到加速器的综述。*ACM Computing Surveys (CSUR)* 54卷，第9期，2021年，1–38页。[2] Lei Bai、Lina Yao、Can Li、Xianzhi Wang 和 Can Wang。2020年。用于交通预测的自适应图卷积循环网络。神经信息处理系统进展 33（2020年），17804–17815页。[3] Zhihao Bai、Zhen Zhang、Yibo Zhu 和 Xin Jin。2020年。{PipeSwitch}：用于深度学习应用的快速流水线化上下文切换。载于

第14届 USENIX 操作系统设计与实现研讨会（OSDI 20）。USENIX协会，美国，499–514页。[4] Nathan Bell 和 Michael Garland。2009年。在面向吞吐量的处理器上实现稀疏矩阵-向量乘法。载于高性能计算、网络、存储与分析会议论文集（俄勒冈州波特兰）（SC '09）。计算机协会，纽约州纽约市，美国，论文第18号，共11页。https://doi.org/10.1145/1654059.1654078[5] Venkatesan T Chakaravarthy、Shiv maran S Pandian、Saurabh Raje、Yogish Sabharwal、Toyotaro Suzumura 和 Shashanka Ubaru。2021年。动态图神经网络的高效扩展。载于高性能计算、网络、存储与分析国际会议论文集。计算机协会，纽约州纽约市，美国，1–15页。[6] Kyunghyun Cho、Bart van Merriënboer、Caglar Gulcehre、Dzmitry Bahdanau、Fethi Bougares、Holger Schwenk 和 Yoshua Bengio。2014年。用于统计机器翻译的 RNN 编码器-解码器短语表示学习。载于2014年计算语言学实证方法会议论文集（EMNLP）。计算语言学协会，卡塔尔多哈，1724–1734页。https://doi.org/ 10 3115 1 14 1179。/v /D - Matthias Fey 和 Jan Eric Lenssen。2019年。[7]使用 PyTorch Geometric 进行快速图表示学习。CoRR abs/1903.02428（2019年），arXiv– 1903 1903 02428 1903 02428。arXiv：。http://arxiv.org/abs/。[8] Alex Fout、Jonathon Byrd、Basir Shariat 和 Asa Ben-Hur。2017年。使用图卷积网络进行蛋白质界面预测。载于《神经信息处理系统进展》，I. Guyon、U. Von Luxburg、S. Bengio、H. Wallach、R. Fergus、S. Vishwanathan 和 R. Garnett（编），第30卷。Curran Associates, Inc.，美国加利福尼亚州长滩，6530–6539页。https://proceedings.neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf[9] Qiang Fu、Yuede Ji 和 H. Howie Huang。2022年。TLPGNN：一种用于图形处理器上图神经网络计算的轻量级两级并行范式。载于第31届高性能并行与分布式计算国际研讨会论文集（美国明尼苏达州明尼阿波利斯）（HPDC '22）。计算机协会，纽约州纽约市，美国，122 134 10 1145 3502181 3531467 –。https://doi.org/。/。[10] 2017 Alberto García-Durán 和 Mathias Niepert。。通过嵌入传播学习图表示。载于《神经信息处理系统进展》第30卷：2017年神经信息处理系统进展年会，2017年12月4日至9日，美国加利福尼亚州长滩。Curran Associates, Inc.，美国，5119–5130页。https://proceedings.neurips.cc/paper/2017/hash/e0688d13958a19e087e123148555e4b4-Abstract.html[11] Alan Gray。2019年。CUDA Graph 入门。https://developer.nvidia.com/blog/cuda-graphs/。[12] Mingyu Guan、Anand Padmanabha Iyer 和 Taesoo Kim。2022年。DynaGraph：大规模动态图神经网络。载于第5届 ACM SIGMOD 图数据管理经验与系统联合国际研讨会（GRADES）和网络数据分析（NDA）论文集（宾夕法尼亚州费城）（GRADES-NDA '22）。计算机协会，纽约州纽约市，美国，论文第6号，共10页。https://doi.org/ 10 1145 3534540 3534691。/。[13] Shengnan Guo、Youfang Lin、Huaiyu Wan、Xiucheng Li 和 Gao Cong。2022年。学习时空图数据的动态特征与异质性以进行交通预测。《IEEE知识与数据工程汇刊》34卷，第11期，2022年，5415–5428页。https://doi.org/。/TKDE 3056502 [14] 1997 Sepp Hochreiter 和 Jürgen Schmidhuber。。长短期记忆。《神经计算》9卷，第8期（1997年），1735–1780页。[15] Guyue Huang、Guohao Dai、Yu Wang 和 Huazhong Yang。2020年。GE-SpMM：面向图神经网络的图形处理器通用稀疏矩阵-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 2020，线上活动 / 美国佐治亚州亚特兰大，11月9日至19日，2020年，Christine Cuicchi、Irene Qualters 和 William T. Kramer（编）。IEEE/ACM，美国，72。https://doi.org/10.1109/SC41405.2020.00076[16] Kezhao Huang、Jidong Zhai、Zhen Zheng、Youngmin Yi 和 Xipeng Shen。2021。理解并弥合当前图神经网络性能优化中的差距。载于第 26 届 ACM SIGPLAN并行编程原理与实践研讨会论文集（线上活动，大韩民国）（PPoPP '21）。计算机协会，纽约州纽约市，美国，119–132。https://doi.org/10.1145/ 3437801.3441585[17] Yanping Huang、Youlong Cheng、Ankur Bapna、Orhan Firat、De- hao Chen、Mia Chen、HyoukJoong Lee、Jiquan Ngiam、Quoc V Le、Yonghui Wu 和 zhifeng Chen。2019。GPipe：利用流水线并行高效训练巨型神经网络。载于《神经信息处理系统进展》。Curran Associates, Inc.，美国，103–112。https://proceedings.neurips.cc/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf[18] Tim Kaler、Nickolas Stathas、Anne Ouyang、Alexandros-Stavros Il- iopoulos、Tao Schardl、Charles E Leiserson 和 Jie Chen。2022。通过快速采样和流水线处理加速图神经网络的训练与推理。《机器学习与系统会议论文集》4（2022），172–189。[19] Seyed Mehran Kazemi、Rishab Goel、Kshitij Jain、Ivan Kobyzev、Akshay Sethi、Peter Forsyth 和 Pascal Poupart。2020。动态图的表示学习：综述。《机器学习研究杂志》21，70（2020），1–73。[20] Thomas N. Kipf 和 Max Welling。2017。基于图卷积网络的半监督分类。载于第 5 届国际学习表征会议（ICLR 2017），2017 年 4 月 24 日至 26 日，法国土伦，会议论文集。OpenReview.net，美国，– 。https: //openreview.net/forum?id=SJU4ayYgl[21] Woosuk Kwon、Gyeong-In Yu、Eunji Jeong 和 Byung-Gon Chun。2020。Nimble：用于深度学习的轻量级并行 GPU 任务调度。《神经信息处理系统进展》33（2020），8343–8354。[22] Haoyang Li 和 Lei Chen。2021。面向动态图的基于缓存的图神经网络系统。载于第 30 届 ACM信息与知识管理国际会议论文集（线上活动，澳大利亚昆士兰）（CIKM '21）。计算机协会，纽约州纽约市，美国，937 946 10 1145 3459637 3482237 – 。[23] https://doi.org/ . / . Shuangchen Li、Dimin Niu、Yuhao Wang、Wei Han、Zhe Zhang、Tian- chan Guan、Yijin Guan、Heng Liu、Linyong Huang、Zhaoyang Du、Fei Xue、Yuanwei Fang、Hongzhong Zheng 和 Yuan Xie。2022。面向大规模分布式图神经网络的超大规模 FPGA即服务架构。载于第 49 届计算机体系结构国际研讨会论文集（纽约，纽约）（ISCA '22）。计算机协会，纽约州纽约市，美国，946–961。https://doi.org/ 10 1145 3470496 3527439 . / . [24] 2013 Justin Luitjens。CUDA 专业技巧：通过向量化内存访问提升性能。https://developer.nvidia.com/blog/cuda-pro-tip-increase-performance-with-vectorized-memory-access/.[25] Chen Ma、Liheng Ma、Yingxue Zhang、Jianing Sun、Xue Liu 和 Mark Coates。2020。用于序列推荐的记忆增强图神经网络。载于第三十四届 AAAI人工智能会议（AAAI 2020）、第三十二届人工智能创新应用会议（IAAI 2020）、第十届 AAAI人工智能教育进展研讨会（EAAI 2020），纽约州纽约市，美国，2020年2月7日至12日。AAAI出版社，美国，5045–5052。https://ojs.aaai.org/index.php/AAAI/article/view/5945[26] Lingxiao Ma、Zhi Yang、Youshan Miao、Jilong Xue、Ming Wu、Lidong Zhou 和 Yafei Dai。2019。NeuGraph：大型图上的并行深度神经网络计算。载于 2019 年 USENIX 年度技术会议（USENIX ATC 2019），美国华盛顿州伦顿，2019年7月10日至12日。USENIX协会，美国，443–458。https://www.usenix.org/conference/atc19/ presentation/ma

[27] Deepak Narayanan、Aaron Harlap、Amar Phanishayee、Vivek Seshadri、Nikhil R. Devanur、Gregory R. Ganger、Phillip B. Gibbons 和 Matei Zaharia。2019。PipeDream：用于深度神经网络训练的通用流水线并行。载于第 27 届 ACM操作系统原理研讨会论文集（美国安大略省亨茨维尔）（SOSP '19）。计算机协会，纽约州纽约市，美国，1–15。https://doi.org/ 10.1145/3341301.3359646[28] Deepak Narayanan、Amar Phanishayee、Kaiyu Shi、Xie Chen 和 Matei Zaharia。2021。内存高效的流水线并行深度神经网络训练。载于第 38 届国际机器学习大会论文集（ICML 2021），2021 年 7 月 18 日至 24 日（《机器学习研究论文集》第 139 卷），Marina Meila 和 Tong Zhang（编）。机器学习研究论文集，美国，7937–7947。http://proceedings.mlr.press/v139/narayanan21a.html [29] 英伟达。2022。NVIDIA：性能分析器用户指南。https://docs.nvidia.com/cuda/profiler-users-guide/. [30] 英伟达。2022。NVIDIA 系统管理接口。https://developer. nvidia.com/nvidia-system-management-interface. [31] Hyungjun Oh、Junyeol Lee、Hyeongju Kim 和 Jiwon Seo。2022。乱序反向传播：一种有效的深度学习调度技术。载于第十七届欧洲计算机系统会议论文集（法国雷恩）（EuroSys '22）。计算机协会，纽约州纽约市，美国，435–452。https://doi.org/10.1145/ 3492321.3519563[32] George Panagopoulos、Giannis Nikolentzos 和 Michalis Vazirgiannis。2021。用于疫情预测的迁移图神经网络。载于第三十五届 AAAI人工智能会议（AAAI 2021）、第三十三届人工智能创新应用会议（IAAI 2021）、第十一届人工智能教育进展研讨会（EAAI 2021），线上活动，2021年2月2日至9日。AAAI出版社，美国，4838–4845。https://ojs.aaai.org/index.php/AAAI/article/view/16616[33] Aldo Pareja、Giacomo Domeniconi、Jie Chen、Tengfei Ma、Toyotaro Suzumura、Hiroki Kanezashi、Tim Kaler、Tao B. Schardl 和 Charles E. Leiserson。2020。EvolveGCN：适用于动态图的演化图卷积网络。载于第三十四届 AAAI人工智能会议（AAAI 2020）、第三十二届人工智能创新应用会议（IAAI 2020）、第十届 AAAI人工智能教育进展研讨会（EAAI 2020），2020年2月7日至12日。AAAI出版社，纽约州纽约市，美国，5363–5370。https://ojs.aaai.org/index.php/AAAI/article/view/5984 [34] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Brad- bury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Kopf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala。2019。PyTorch：一种命令式风格的高性能深度学习库。载于《神经信息处理系统进展》第 32 卷。Curran As- sociates, Inc.，美国，8024–8035。https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf [35] PyTorch。2022。PyTorch：PyTorch Profiler。https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html。[36] Ryan A. Rossi 和 Nesreen K. Ahmed。2014。NetworkRepository：一个带有可视化交互分析功能的图数据仓库。CoRR abs/1410.3560（2014），arxiv–1410。arXiv:1410.3560 http://arxiv.org/ abs/1410.3560[37] Ryan A. Rossi 和 Nesreen K. Ahmed。2015。带有交互式图分析和可视化功能的网络数据仓库。载于第二十九届 AAAI人工智能会议论文集，2015年1月25日至30日，美国得克萨斯州奥斯汀，Blai Bonet 和 Sven Koenig（编）。AAAI出版社，美国，4292–4293。http://www.aaai.org/ocs/index.php/ AAAI/AAAI15 9553 /paper/view/ Benedek Rozemberczki、[38] Paul Scherer、Yixuan He、George Panagopou- los、Alexander Riedel、Maria Astefanoaei、Oliver Kiss、Ferenc Beres、Guzmán López、Nicolas Collignon 和 Rik Sarkar。2021。PyTorch 几何时序：使用神经机器学习模型进行时空信号处理。载于第30届ACM信息与知识管理国际会议论文集（线上活动，昆士兰，澳大利亚）（CIKM '21）。计算机协会，纽约州纽约市，美国，4564–4573。https://doi.org/10.1145/ 3459637.3482014[39] Joakim Skarding、Bogdan Gabrys 和 Katarzyna Musial。2021。使用动态图神经网络构建和建模动态网络：综述。《IEEE Access》9（2021），79143–79168。[40] Xinkai Song、Tian Zhi、Zhe Fan、Zhenxing Zhang、Xi Zeng、Wei Li、Xing Hu、Zidong Du、Qi Guo 和 Yunji Chen。2021。Cambricon-G：一种多功能节能加速器，适用于动态图神经网络。《IEEE集成电路与系统计算机辅助设计汇刊》41，1（2021），116–128。[41] Qidong Su、Minjie Wang、Da Zheng 和 Zheng Zhang。2021。并行图神经网络训练的自适应负载均衡。载于MLSys图神经网络与系统研讨会论文集（GNNSys）。mlsys.org，美国加利福尼亚州圣何塞，1–8。[42] Qingxiao Sun、Yi Liu、Hailong Yang、Ruizhe Zhang、Ming Dun、Mingzhen Li、Xiaoyan Liu、Wencong Xiao、Yong Li、Zhongzhi Luan 和 Depei Qian。2022。CoGNN：面向图形处理器上并发图神经网络训练的高效调度。载于国际高性能计算、网络、存储与分析会议论文集（得克萨斯州达拉斯）（SC '22 39, 15）。IEEE出版社，洛斯阿拉米托斯，美国，文章页码。[43] John Thorpe、Yifan Qiao、Jonathan Eyolfson、Shen Teng、Guanzhou Hu、Zhihao Jia、Jinliang Wei、Keval Vora、Ravi Netravali、Miryung Kim 和 Guoqing Harry Xu。2021。Dorylus：使用分布式CPU服务器和无服务器线程实现经济、可扩展且准确的图神经网络训练。载于第15届USENIX操作系统设计与实现研讨会（OSDI 2021），2021年7月14日至16日。USENIX协会，美国，495–514。https://www.usenix.org/conference/osdi21/presentation/ thorpe[44] Petar Velickovic、Guillem Cucurull、Arantxa Casanova、Adriana Romero、Pietro Liò 和 Yoshua Bengio。2018。图注意力网络。载于第六届国际学习表征会议（ICLR 2018），加拿大不列颠哥伦比亚省温哥华，2018年4月30日至5月3日，会议论文集。OpenReview.net，美国，1–12。https://openreview. net/forum?id=rJXMpikCZ [45] Cheng Wan、Youjie Li、Cameron R. Wolfe、Anastasios Kyrillidis、Nam Sung Kim 和 Yingyan Lin。2022。PipeGCN：通过流水线特征通信实现图卷积网络的高效全图训练。载于第十届国际学习表征会议（ICLR 2022），线上活动，2022年4月25日至29日。OpenReview.net，美国，1–24。https://openreview.net/forum?id=kSwqMH0zn1F[46] Lei Wang、Qiang Yin、Chao Tian、Jianbang Yang、Rong Chen、Wenyuan Yu、Zihang Yao 和 Jingren Zhou。2021。FlexGraph：用于图神经网络训练的灵活高效分布式框架。载于第十六届欧洲计算机系统会议论文集（线上活动，英国）（EuroSys '21）。计算机协会，纽约州纽约市，美国，67–82。https://doi.org/10.1145/3447786.3456229[47] Minjie Wang、Lingfan Yu、Da Zheng、Quan Gan、Yu Gai、Zihao Ye、Mufei Li、Jinjing Zhou、Qi Huang、Chao Ma、Ziyue Huang、Qipeng Guo、Hao Zhang、Haibin Lin、Junbo Zhao、Jinyang Li、Alexander J. Smola 和 Zheng Zhang。2019。深度图学习库：迈向高效、可扩展的图上深度学习。CoRR abs/1909.01315（2019），arxiv–1909。arXiv:1909.01315 http://arxiv.org/abs/1909.01315[48] Yuke Wang、Boyuan Feng、Gushu Li、Shuangchen Li、Lei Deng、Yuan Xie 和 Yufei Ding。2021。GNNAdvisor：一种用于图形处理器上图神经网络加速的自适应高效运行时系统。载于第15届USENIX操作系统设计与实现研讨会（OSDI 2021），2021年7月14日至16日，Angela Demke Brown 和 Jay R. Lorch（编）。USENIX协会，美国，515–531。https://www.usenix.org/conference/osdi21/presentation/wang-yuke

[49] Jianbang Yang、Dahai Tang、Xiaoniu Song、Lei Wang、Qiang Yin、Rong Chen、Wenyuan Yu 和 Jingren Zhou。2022。GNNLab：一种面向图形处理器上基于采样的图神经网络训练的分解式系统。载于第十七届欧洲计算机系统会议论文集（法国雷恩）（*EuroSys '22*）。计算机协会，纽约州纽约市，美国，417–434。https://doi.org/10.1145/3492321.3519557[50] Ling Zhao、Yujiao Song、Chao Zhang、Yu Liu、Pu Wang、Tao Lin、Min Deng 和 Haifeng Li。2019。T-GCN：用于交通预测的时序图卷积网络。*IEEE*智能交通系统汇刊21，9（2019），3848–3858。

[51] Hongkuan Zhou、Da Zheng、Israt Nisa、Vasileios Ioannidis、Xi- ang Song 和 George Karypis。2022年。TGL：一个用于十亿规模图时序图神经网络训练的通用框架。*CoRR* abs/2203.14883 (2022年), arxiv–2203. https://doi.org/10.48550/arXiv.2203. 14883 arXiv:2203.14883[52] Rong Zhu、Kun Zhao、Hongxia Yang、Wei Lin、Chang Zhou、Baole Ai、Yong Li 和 Jingren Zhou。2019年。AliGraph：一个全面的图神经网络平台。*VLDB*基金会会刊 12, 12 (2019年), 2094–2105。