

基于概率邻域扩展分析与缓存的通信高效图神经网络

Tim Kaler [*1 2] **Alexandros-Stavros Iliopoulos** [*1 2] **Philip Murzynowski** [*1 2] **Tao B. Schardl** [1 2] **Charles E. Leiserson** [1 2] **Jie Chen** [2 3]

arXiv:2305.03152v1 [cs.LG] 4 May 2023

摘要

自图神经网络（GNN）问世以来，针对大规模图的图神经网络训练与推理便一直受到广泛研究，这得益于图神经网络在推荐系统和金融取证等应用中的广泛使用与成功。本文关注在分布式环境下采用逐节点采样的图神经网络小批量训练与推理。在这种环境中，顶点特征必须分布式存储，因此特征通信会成为阻碍可扩展性的主要瓶颈。为在不影响预测准确率的前提下大幅减少通信量，我们提出了一种缓存策略，用于缓存远程分区中与频繁访问的顶点相关联的数据。该策略基于对多跳邻域采样过程中顶点级包含概率（VIP）的分析；多跳邻域采样可能使邻域扩展到远超图的分区边界。VIP 分析不仅能够消除通信瓶颈，还提供了一种组织内存数据的方法，可优先将 GPU存储分配给访问频率最高的顶点特征。我们推出了 SALIENT++，它在先前最先进的 SALIENT 系统基础上进行了扩展，使其能够处理分区特征数据，并利用 VIP 驱动的缓存策略。SALIENT++ 通过采用深度流水线，在保留 SALIENT 本地训练效率和可扩展性的同时，大幅减少通信量，且仅消耗 SALIENT 所需存储空间的一小部分。我们给出了基于 Open Graph Benchmark 数据集的实验结果，并证明：在 8 台单GPU机器上使用 SALIENT++训练 3 层 GraphSAGE 模型，比在 1 台单GPU机器上使用 SALIENT 快 7.1×，也比在 8 台单GPU机器上使用 DistDGL 快 12.7×。

1 引言

图神经网络（GNN）是一类重要的机器学习模型，它们融入关系归纳偏置，以便在图结构数据上有效地进行表示学习（Li 等人，2016；Kipf 和 Welling，2017；Hamilton 等人，2017；Veli$\check{}$ckovi$\acute{}$c 等人，2018；Xu 等人，2019）。这些神经网络已成功应用于多种场景，包括产品推荐、交通预测和金融取证（Ying 等人，2018；Li 等人，2018；Weber 等人，2019）。在许多应用中，数据持续收集，生成的图迅速增长，因此需要高效的图神经网络训练和推理系统，以应对图数据的爆炸式增长。

本文研究了在分布式环境下，采用邻域采样进行小批次训练和推理的情形，其中顶点数据（特征）被分区存储在多台机器上。作为与全批量优化相对，小批量优化是训练神经网络的典型方式，但由于网络层之间邻域规模呈指数增长，这给图神经网络带来了独特的挑战（Chen 等人，2018）。邻域采样是缓解这一问题的一种流行且有效的方法（Hamilton 等人，2017；Chen 等人，2018；Ying 等人，2018；Zou 等人，2019；Zeng 等人，2020；Ramezani 等人，2020；Dong 等人，2021）。在分布式图神经网络训练中，每台机器都会为属于其分区的顶点小批量计算训练损失。在训练优化的每一步中，一组按分区划分的小批次构成一个“分布式小批量”，用于更新图神经网络模型参数。分布式图神经网络推理的组织方式类似。

即使采用邻域采样，小批量邻域扩展仍会造成通信瓶颈，因为每台机器都需要访问采样顶点的远程数据。由于小批次和邻域采样具有随机性，通信模式也具有随机性。通信时间往往超过训练计算所需的时间。表格中展示了这种瓶颈效应的一个示例1，我们稍后将对此进行说明。

[*]共同贡献 [1]MIT计算机科学与人工智能实验室 [2]MIT-IBM Watson人工智能实验室 [3]IBM研究院。通讯作者：Jie Chen <chen-jie@us.ibm.com>。

MLSys 会议论文集，6^{th} *MLSys* 会议，迈阿密海滩，美国佛罗里达州，2023 年。版权所有 © 2023，作者所有。

为克服通信瓶颈，我们提出通过顶点纳入概率分析邻域访问模式，并基于该分析制定缓存策略。我们将此分析称为“VIP 分析”。不同于常用的顶点访问概率估算启发式方法——例如基于顶点度和扩展边界前沿（Lin 等人，2020）、随机游走（Dong 等人，2021；Min 等人，2021）或模拟 GNN 计算（Yang 等人，2022）——VIP 分析基于对实际随机邻域扩展过程的解析模型，估算所有图顶点的访问概率。我们为重要的逐节点采样方案类别推导了具体的 VIP 模型（Hamilton 等人，2017；Ying 等人，2018；Veli ˇckovi ´c 等人，2018；Xu 等人，2019；Liu 等人，2020），并发现，由此得到的缓存策略能够大幅减少采用逐节点采样的分布式图神经网络中的通信量，同时仅带来适度的内存开销。此外，VIP 分析还提供了一种组织内存数据的方法：优先将访问最频繁的顶点特征存储在 GPU 中，从而减少主机到设备数据传输。

我们的系统 SALIENT++，基于 SALIENT（Kaler 等人，2022）开发。SALIENT 是一种高效的图神经网络系统，通过经过性能优化的邻域采样、共享内存并行批次准备和数据传输流水线化，实现了最先进的性能。SALIENT 的每轮耗时几乎等于仅执行模型计算所需的 GPU 耗时（实际上隐藏了小批次构建、数据传输和梯度通信的开销），并且能够很好地扩展到更多机器上。但 SALIENT 的缺点是会在每台机器上复制整个数据集，从而对数据集大小施加了严格限制。要扩展 SALIENT 以分布式存储顶点特征数据，就必须解决特征通信瓶颈。

SALIENT++ 采用 VIP 分析驱动的缓存策略和深度流水线，实现了可扩展性与高效率。它仅需 SALIENT 一小部分的内存，就能达到与 SALIENT 几乎相当的性能。为凸显 SALIENT++的成效，表1 列出了在 SALIENT 基础上逐步改进后所取得的性能。此示例使用 ogbn-papers100M 数据集（Hu 等人，2020），并采用 METIS（Karypis 和 Kumar，1997）对图进行分区，以边割最小化为目标，并对每个分区中的训练、验证及总顶点数以及边总数设置平衡约束。与表1相配的图1通过简化的计算概况和存储需求示意图，展示了 SALIENT 与 SALIENT 后续优化之间的关键差异++。

我们首先探究 SALIENT 在机器间对特征数据进行不相交分区时的分布式性能。我们修改 SALIENT，使其仅复制图结构，并按需批量通信每个小批次及其采样邻域中的顶点特征。

表*1*。在无向 ogbn-papers100M 数据集上，采用 3 层 GraphSAGE架构、采样扇出数（15,10,5）和 256 维隐藏层，逐步提升复杂度的分布式图神经网络训练系统的每轮运行时间。对于采用远程特征缓存的系统，缓存大小占每台机器本地顶点特征大小的比例分别为 8%（2 台机器）、16%（4 台机器）或 32%（8 台机器）。

	机器数量			
系统	1	2	4	8
SALIENT（完全复制）	20.7秒	10.76秒	6.02秒	3.08秒
+ 分区特征	—	33.04秒	15.98秒	10.85秒
+流水线通信	—	16.12秒	8.73秒	5.43秒
+ 特征缓存	—	10.51秒	5.45秒	2.91秒

而在机器间分布顶点特征，虽然大幅降低了 SALIENT 的内存需求，却也立即导致性能下降。具体而言，与完全复制的 SALIENT 相比，我们观察到在两台或四台机器上速度约降低 2–3× ，在八台机器上则约降低3.5× 。这种速度下降表现为 GNN 模型计算之间的高延迟（图 1第 2 行）。通过流水线化，将特征通信与其他 GNN 操作（采样、本地特征切片、主机到设备的数据传输、训练计算和模型更新）重叠执行，可以带来一定改善，但系统仍受通信瓶颈限制（图1第 3 行）。

不过，如果允许少量内存开销，将频繁访问的远程特征缓存在本地，减少通信量与流水线化的综合效果便能大幅提升性能。缓存带来的通信量大幅减少，使流水线化能够（几乎）完全隐藏通信（图1第 4 行）。在 K 台机器上，SALIENT++ 将 SALIENT 的特征总内存占用减少了近 K 倍，同时达到与 SALIENT 的完全复制策略相当的性能。

总之，本研究的贡献如下：

1. 分析采用逐节点采样进行图神经网络邻域扩展时的顶点纳入概率（VIP）。我们展示了如何利用 VIP 分析，最大限度地减少机器之间的预期通信量，以及每台机器内部的主机到设备数据传输。

2. SALIENT++ 的设计与实现：这是一个高效且可扩展的分布式图神经网络训练与推理系统。该系统基于两个关键组件：一种基于 VIP 分析、针对远程和局部特征的最大似然静态缓存策略；以及一条深度流水线，可将特征通信与其他图神经网络操作重叠执行，从而确保高利用率

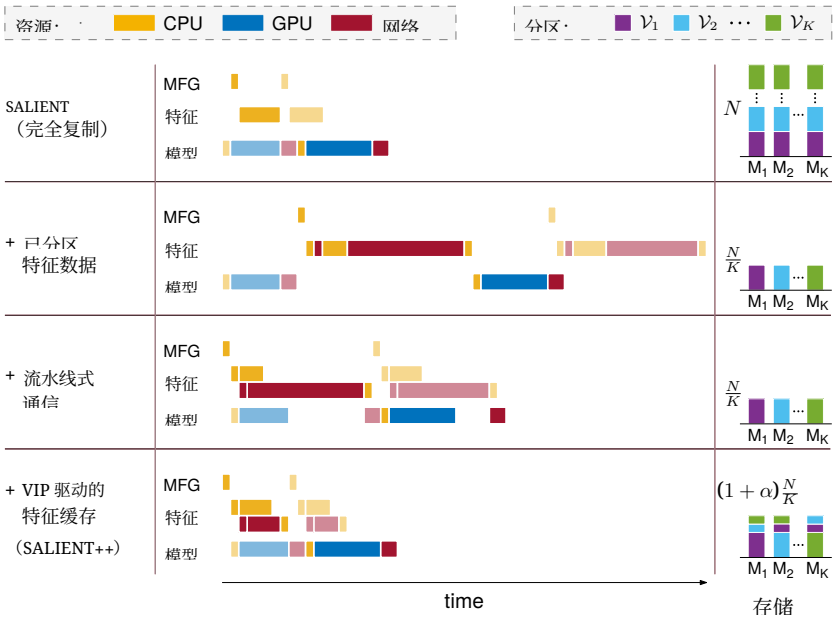


图*1*。分布式图计算概况和特征数据存储示意图，展示前身系统 SALIENT 及后续优化，最终形成我们的系统 SALIENT++。**左侧：**一台机器中系统资源（CPU、GPU 和网络）的利用率。每个小批次的处理分为三个部分。“MFG”：消息流图构建。“特征”：局部特征张量切片、特征通信与合并，以及主机到设备传输。“模型”：前向和反向传播，以及机器之间用于模型更新的通信。除突出显示的小批次外，概况图还以淡色显示前一个和后一个小批次的部分内容，以展示流水线操作之间的重叠。**右侧：**颜色堆叠条表示 K 台机器中每台机器本地存储的特征向量数量。

可用网络带宽。

3. 在三个大型基准图上，对 SALIENT++ 的端到端训练性能和可扩展性进行了系统评估。在这些数据集中规模最大的数据集无法完全存储在单台机器的内存中。随着单GPU机器数量从 4 台扩展至 8 台，SALIENT++ 实现了 $1.75\times$ 的加速比；当单GPU机器数量进一步从 8 台扩展至 16 台时，又额外实现了 $1.45\times$ 的加速比。

2 背景与相关工作

本节简要回顾背景知识，并介绍图神经网络及其训练/推理的相关符号表示。还将讨论系统和分布式训练方面的相关工作。

2.1 图神经网络

本文聚焦于消息传递神经网络（MPNNs）（Gilmer等，2017年），该类网络涵盖了许多在大规模图中备受关注的图神经网络。设 $G=(\mathcal{V},\mathcal{E})$ 为一个图，其中 $\mathcal{V}$ 为顶点集，$\mathcal{E}$ 为边集。设 $X\in\mathbb{R}^{N\times D}$ 为顶点特征矩阵，其中 N 为顶点数量，D 为特征维度。X的一行记为 $x_v\in\mathbb{R}^D$，是顶点 v的特征向量。令 $\ell=0,\ldots,L$ 表示层索引，且 $\mathcal{N}_1(v)$ 为 v的一跳邻域。MPNNs 使用以下更新规则来定义一层：

$$h_v^{\ell}=\mathrm{UPD}^{\ell}\Big(h_v^{\ell-1},\mathrm{AGG}^{\ell}\big(\{h_u^{\ell-1}\mid u\in\mathcal{N}_1(v)\}\big)\Big),\quad(1)$$

其中，h_v^{ℓ} 是 v的第ℓ 层表示，AGG^{ℓ} 是集合聚合函数，UPD^{ℓ} 是二元更新函数，而 $h_v^0=x_v$。可以看出，一个顶点的表示 v 取决于 v 的前层表示及其在 $\mathcal{N}_1(v)$中的邻居节点；这种依赖是递归的。

图神经网络在（1）中两个函数的设计上各不相同。例如，在 GraphSAGE（Hamilton 等人，2017）中，AGG^{ℓ} 是均值、LSTM 或池化算子，而 UPD^{ℓ} 将两个参数拼接起来，并应用一个线性层。在 GIN（Xu 等人，2019）中，AGG^{ℓ} 是 $\{h_u^{\ell-1}\}$ 的求和，而 UPD^{ℓ} 则将其参数相加，再经过一个多层感知机。在 GAT（Veli˘ckovi´c 等人，2018）中，AGG^{ℓ} 是恒等映射，而 UPD^{ℓ} 会将 $W^{\ell-1}h_u^{\ell-1}$ 对所有 $u\in\{v\}\cup\mathcal{N}_1(v)$ 进行加权组合，从而计算出 h_v^{ℓ}；其中权重为注意力系数，$W^{\ell-1}$ 是该层的参数矩阵。

2.2 小批量训练

神经网络可以通过梯度下降（全批量训练）或随机梯度下降方法（小批量训练）进行训练。大多数深度学习文献都主张采用小批量训练，理论和实证证据表明，这种方法收敛更快、泛化能力更强，也更适合面向 GPU 的计算基础设施（Bottou 等人，2018）。

小批量训练给图神经网络带来了独特的挑战，通常称为“邻域爆炸”问题：要计算顶点 v的训练损失，公式（1）表明，需要递归地从邻域中获取信息，其规模会随着层数增加而呈指数级增长。对于规模合理的小批次，多跳邻域很快就会覆盖图中的很大一部分，带来高昂的时间成本，甚至抵消收敛速度提升所节省的时间。此外，存储需求很容易超过 GPU 的内存容量。

2.3 邻域采样

通过采样限制邻域规模，是解决邻域爆炸问题的有效方法。主要有三类采样策略：逐节点采样（Hamilton 等人，2017；Ying 等人，2018）、逐层采样（Chen 等人，2018；Zou 等人，2019）和子图采样（Chiang 等人，2019；Zeng 等人，2020）。

逐节点采样方法会针对每个顶点分别调整其邻域 v，随机选取一个最多包含 f 个邻居节点的子集（称为扇出）。逐层采样方法会收集一个小批次中所有顶点的邻居节点，并从这些邻居节点的并集中采样邻域。这种采样按层递归进行，并可使用非平凡的采样分布，在保持无偏性的同时控制预激活方差。非线性激活函数无论如何都会破坏无偏性，但仍可基于渐近一致性建立训练收敛结果（Chen 和 Luss，2018）。子图采样方法会采样一个连通子图，并计算仅限于该子图的小批次损失。本文关注应用广泛的逐节点采样方法。

2.4 小批次推理

与训练类似，推理既可以采用全批次，也可以采用小批次。全批次训练还是小批量训练的选择，主要取决于收敛和泛化；而推理方式的选择则取决于计算模式和实现工作量。全批次推理无需进行邻域采样，但需要针对不同的 GNN 架构分别实现前向传播。另一方面，小批次推理可以复用训练时的前向函数，从而提高应用开发者和架构设计者的工作效率；但邻域采样的随机性可能导致返回不同的预测结果，且无法保证其准确率与不进行采样时相同。尽管如此，SALIENT 的作者通过实验证明，合理选择扇出数不会影响预测准确率（Kaler 等人，2022）。本文采用小批次推理。

2.5 相关工作

由于邻域聚合的计算模式，大规模图上的图神经网络训练系统在设计和实现上与常规神经网络训练系统有很大差异。为简化设计，许多图神经网络系统基于全批次训练开发，从而避免了邻域采样的复杂性。相关示例包括 NeuGraph（Ma 等人，2019）、Roc（Jia 等人，2020）、DeepGalois（Hoang 等人，2021）、FlexGraph（Wang 等人，2021a）、Seastar（Wu 等人，2021）、GNNAdvisor（Wang 等人，2021b), DistGNN(Md 等，2021），以及 BNS-GCN（Wan 等，2022）。其中一些系统基于常见的深度学习框架构建，例如 TensorFlow（Abadi 等，2015）和 PyTorch（Paszke 等，2019），而另一些则基于自主开发的编程模型构建。

执行小批量训练的系统示例包括 DistDGL（Zheng 等人，2020）、Zero-Copy（Min 等人，2021）、GNS（Dong 等人，2021）以及 P^3（Gandhi 和 Iyer，2021）。然而，这些出版物报告的结果要么是在仅配备 CPU 的多台机器上获得的，要么是在配备一个或多个 GPU 的单台机器上获得的。较新版本的 DistDGL（DistDGLv2（Zheng 等人，2021））、SALIENT（Kaler 等人，2022）以及我们的系统 SALIENT++ 展示了分布式多 GPU 环境下的结果。

SALIENT++ 采用边切图划分，将数据分布到各台机器上。DistGNN（Md 等人，2021）采用了另一种方法，即顶点切分，该方法确保每条边都局部存储在某台机器上。这种方法的缺点是，一个顶点可能会被分配到多台机器，从而由于特征复制而产生内存开销；据报道，这一开销最高可达 5×。另一方面，++SALIENT 在我们的实验中通过缓存实现了较高的分布式性能，同时内存开销低于 50%。

SALIENT++采用缓存策略来减少通信和数据传输。我们提出了一种基于邻域采样期间顶点访问概率分析的策略。尽管已有其他系统在这一背景下探索了缓存，但以往的缓存策略都是启发式的，使用顶点度（Lin et al.，2020）、随机游走（Donget al.，2021；Min et al.，2021）、或模拟访问频率（Yangetal.，2022）等代理指标。DGCL（Cai et al.，2021采用了另一种减少通信时间的方法，通过通信规划算法针对特定网络拓扑优化通信模式。

Marius 和 MariusGNN（Mohoney et al.，2021；Waleffeetal.，2022）是利用外部存储在单台机器上运行的核外训练系统。它们运行在与我们不同的环境中，但我们的顶点纳入概率分析可作为补充，用于优化磁盘上的数据表示和磁盘 I/O。

3 图神经网络节点级采样的顶点纳入概率

本节介绍一种有原则的方法，用于估算并降低采用逐节点采样的分布式图神经网络中的数据访问成本。逐节点采样最早由 GraphSAGE（Hamilton 等人，2017）引入，并应用于多种 GNN 架构（Ying 等人，2018；Veli˘ckovi´c 等人，2018；Xu 等人，2019；Liu 等人，2020）。我们

首先介绍顶点包含概率（***VIP***）分析，以计算任意顶点出现在某个分区中小批次的 L 跳扩展邻域内的概率。VIP 分析考虑了图神经网络在小批次和逐节点采样场景下邻居访问的独特结构。随后我们将展示，VIP 分析能够实现一种简单但极为有效的策略，以减少 GNN 计算过程中的数据移动。

本节推导的用于逐节点采样的 VIP模型不适用于其他采样方案，例如逐层采样或子图采样。或许可以用类似的方法为这些其他采样方案推导 VIP模型，但这超出了我们的研究范围。

3.1 使用逐节点采样分析 L 跳邻域中的顶点纳入概率

我们分析以下随机过程，该过程对采用节点级采样的 GNN 架构中的邻域扩展进行建模。(i) 从一组顶点开始，而不是从单个顶点开始。(ii) 对起始集合中的每个顶点，从其直接邻居中无放回地采样一组邻居节点，从而扩展遍历前沿。(iii) 将采样得到的邻域的并集作为新的起始集合，重复步骤 (ii)。该过程在重复 L 次后终止（即经过相应跳数）。我们假设采样在不同跳数之间以及当前跳数集合中的不同顶点之间相互独立。也就是说，尽管每个顶点都是从其直接邻居中无放回采样，但不同顶点可能采样到同一个直接邻居。

我们希望估计按照上述随机过程得到的扩展邻域中的顶点纳入概率，也称为 VIP 值。VIP 值构成一个概率向量 $p(u)$，涵盖所有图顶点。我们可以通过逐跳 VIP 向量 $p^{[h]}(u)$来表示 $p(u)$，其中包含任意顶点 u 在距离起始集合恰好 h跳处被采样的概率。（跳数为-h的邻域包含第 ℓ 个 GNN 层的输入顶点，$\ell = L - h$。）只有当 u 的每个邻居节点 $v \in \mathcal{N}_1(u)$都满足以下条件之一时，顶点 u 才不会在第 h 跳被采样：v 未在第 $h-1$ 跳被采样；或者 v 已被采样，但没有从其邻居节点中选中 u。以下命题说明如何在图中传播初始集合的概率，以计算 VIP 值。证明见附录 C。

命题1 (采用逐节点采样进行小批次邻域扩展时的顶点纳入概率)。 顶点 $u \in \mathcal{V}$ 出现在任意小批次的逐节点采样 L 跳扩展邻域中的概率为

$$p(u) = 1 - \prod_{h=1}^{L} \big(1 - p^{[h]}(u)\big), \tag{2}$$

$$p^{[h]}(u) = 1 - \prod_{v \in \mathcal{N}_1(u)} \big(1 - t_h(u,v)\, p^{[h-1]}(v)\big), \tag{3}$$

其中，$p^{[0]}(v)$ 表示 v 位于小批次中的概率，而 $t_h(u,v)$ 表示在第 $h = 1$ 跳时，v 将 u 采样为邻居的转移概率，……，L。

命题1的VIP模型适用于逐节点采样中的任意初始采样方式和逐跳转移概率函数。例如，如果像GraphSAGE中那样，以均匀随机且不放回的方式对小批次和逐顶点邻居节点进行采样，则有：$p^{[0]}(u) = B/|\mathcal{T}|$，当 $u \in \mathcal{T}$时成立，否则为0，其中 B 为小批量大小，$\mathcal{T}$为训练顶点集合；并且$t_h(u,v) = \min\{1, f_h/d(v)\}$，当 $u \in \mathcal{N}_1(v)$时成立，否则为0，其中 $d(v)$为 v的（出）度。非均匀邻居节点采样模型可通过相应的转移概率矩阵或矩阵组来实现。

邻域扩展随机过程在随机游走和完整邻域扩展之间进行了连续参数化。如果初始集合大小和逐层扇出数均等于 1，则该过程成为随机游走。如果扇出数大于图顶点的最大入度，则该过程成为完整邻域扩展。上述两种特殊情况下的 VIP模型是线性的，而命题 1 中的广义模型是非线性的；尽管如此，这些模型的计算复杂度均为：$O(L(M+N))$。

3.2 顶点特征缓存

现在，我们将展示如何将 VIP 分析应用于缓存策略，以最小化分布式机器之间的通信以及每台机器中的主机到设备数据传输。

通信量削减 我们考虑最大似然意义下的最优静态缓存策略，以减少分区间分布式图神经网络中的通信量。该策略十分直接：每台机器都将最有可能被其访问的远程特征副本添加到本地图顶点特征存储中，从而最小化通信量总期望值。对于给定的初始分区——且不对小批次和邻接顶点的采样顺序作任何假设——该策略与 VIP 分析直接相关。具体而言，我们根据相应的初始概率 $p_k^{[0]}(u)$ 计算分区级 VIP 值 $p_k(u)$。随后，通过按照递减 $p_k(u)$ 的顺序对远程顶点进行排序，确定第 k 个分区的缓存内容。

我们用相应的复制因子 α来衡量远程特征缓存的大小。复制因子的定义是，每台机器中存储的缓存特征向量数量为 $\alpha N/K$，其中 K 是分区或机器的数量。也就是说，每个分区中缓存顶点与本地顶点的比例为 α，平均而言，每个特征向量存储在 $(1+\alpha)$ 台机器中。我们可以说，一个

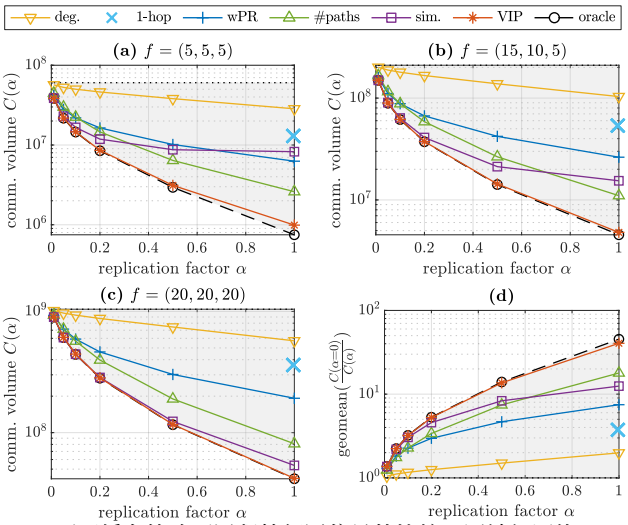


图 2. 不同缓存策略下远程特征通信量的比较。图神经网络：3 层 GraphSAGE，变化的扇出数 f，小批量大小 1024。数据集：ogbn-papers100M（无向）。分区：采用 METIS 进行 8 路划分，以边割为目标，并对边以及训练/验证/测试顶点进行均衡。**(a)–(c)** 100 轮中每轮的平均通信量，以顶点数计（对数刻度）。越低越好。通信量上界（无缓存）与下界（预言机缓存）之间的可行区域以灰色阴影表示。**(d)** 相对于无缓存，跨不同扇出数的几何平均改进（对数刻度）。越高越好。

分区或缓存复制一个远程顶点，指的是它复制了相应的顶点特征数据。

图 2 比较了多种静态特征缓存策略，并证明命题 1 中的分析型 VIP 模型能够有效地得出最优策略。具体比较的策略如下：“deg.” 按度对顶点排序，并过滤掉从分区训练集无法到达的远程顶点（Lin 等人，2020）；“1-hop” 为每个分区复制整个 1 跳邻域；“wPR” 按顶点经过 5 轮加权反向 PageRank 模型迭代后的得分排序（Min 等人，2021），阻尼因子为 0.85；“#paths” 按路径数量对顶点排序，这些路径的长度为 $\leq L$，且从任一本地训练顶点出发并到达这些顶点；“sim.” 按经验 VIP 估计值对顶点排序，通过统计 2 个模拟训练轮次中逐顶点的访问频率来测量（Yang 等人，2022）；“VIP” 按命题 1中的 VIP 值对顶点排序；“oracle” 则在训练后根据顶点的实际访问频率对其追溯排序，从而给出通信量的下界。[1] 我们得出以下观察结果。

[1]对于所有缓存策略，我们都会针对每个分区计算顶点排名。这与这些策略最初引入时的设置不同：在原始设置中，顶点缓存依据的是单一的全局排名分数。

• 最优性。 分析型 VIP 策略在不同扇出数和复制因子下均能实现接近最优的通信量（与“oracle”的差距在 5% 以内）。对于 $f = (5, 5, 5)$ 和 $\alpha = 1.0$，差距较大（~30%），这是因为排名较低的顶点访问情况存在方差；随着样本增多（例如扇出数或轮次增加），差距会缩小。

• 通信降低效果。 VIP 驱动缓存对于降低通信量非常有效。与不使用缓存相比，VIP 驱动策略在较小复制因子 $\alpha \in [0.05, 0.20]$ 下实现了 2.2×–5.3× 的几何平均降低幅度，而当复制因子超过 0.50 时，降低幅度超过 10× 。与其他缓存策略相比，该策略始终能够实现更低的通信量，并且其相对改进幅度会随复制因子增加而提高。

• 局部信息并不充分。 高阶节点和一跳光环策略未考虑邻域扩展，因此相比不使用缓存只能带来有限的改进。缓存多跳邻居将进一步降低通信量（Lin 等人，2020），但也会使复制因子急剧增大。

• 针对扩展过程调整模型的影响。VIP 驱动策略的效果最高可比“wPR”和“#paths”策略分别高出 4× 和 2× 。这两种策略都考虑了图结构扩展，但“wPR”未考虑 GNN 扇出和层数，而“#paths”未直接考虑采样。

• 经验估计的优势与局限。对于较低的复制因子和较大的扇出数，经验 VIP 估计值（分别对应“sim.”和“VIP”策略）与分析 VIP 估计值的结果相近；但随着复制因子增大或扇出数减小，经验估计的效果会变差。对于复制因子 0.50 和 1.0，分析策略相较于经验策略的相对总体改进分别为 1.6× 和 3.2×。经验方法的优点是适用于任何采样方案，但要准确估计访问频率较低的顶点的 VIP 值，也需要越来越多的样本。

减少主机到设备的数据传输 VIP 分析还可用于减少每台机器上局部特征的主机到设备数据传输量。例如，假设每个 GPU 在整个 GNN 计算期间（即跨越所有小批次）都能在内存中保留一部分局部特征数据。我们可以根据 $p_k(u)$ 对 k 分区中的局部顶点进行排名，并将排名最高的顶点特征保留在 GPU 上。与远程顶点缓存一样，该策略可最大限度地减少每个分区中局部顶点所导致的预期数据传输量。

4 SALIENT++：通过缓存和流水线化实现快速、可扩展的分布式图神经网络训练

本节介绍了 SALIENT++ 的设计。SALIENT++ 是一个快速且可扩展的系统，用于在大型分区数据集上执行图神经网络的分布式小批量训练。SALIENT++ 的设计由三个主要组件构成：

顶点重排序 一种基于图分区和 VIP 值的顶点排序，可在 CPU 和 GPU 之间高效地对子分区中的本地特征数据进行划分，从而减少主机到设备的数据传输。

数据复制 一种经济的数据复制策略，利用 VIP 分析减少获取远程特征时的通信量，同时仅需少量额外内存。

流水线通信 一种流水线化的分布式小批次准备系统，可隐藏采样、数据传输以及远程特征数据的机器间通信所产生的延迟。

这三种技术相结合，使 SALIENT++ 能够在分区后的数据图上执行图神经网络训练，同时达到高度优化的图神经网络训练系统的效率水平。此类系统通过在所有机器之间完全复制顶点特征数据来执行分布式计算。因此，SALIENT++ 能够在完全复制不可行的大规模数据集上高效地执行图神经网络训练。

正如我们将在第5节的实证评估中讨论的那样，SALIENT++ 即使在硬件配置较为普通的集群中运行，也能在大型数据集上实现高性能的图神经网络训练。

4.1 顶点特征的分区与重排序

下面介绍 SALIENT 如何对图的顶点特征进行分区，以减少机器间通信和主机到设备的通信。SALIENT++ 采用两级策略，在机器和设备之间分配顶点特征，如图3所示。顶层根据图的顶点分区来分配顶点特征，底层则将每个分区的本地顶点特征划分到 GPU 和 CPU 内存中。

SALIENT++处理使用图划分算法（如 METIS）划分的图数据集（Karypis & Kumar，1997）。生成图分区时，会确保训练、验证和测试顶点均衡。作为平衡每个分区工作量的启发式方法，
此外，还使用一项标准来平衡每个分区中的边数。每台机器都会分配到训练/验证/测试顶点的一个子集，并根据该机器的分区在本地存储顶点特征数据。

对图重新排序，使得：(a) 分区内顶点的索引

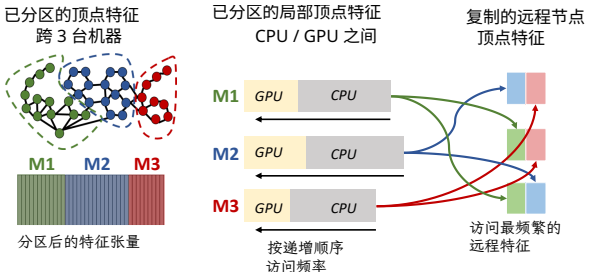


图 3。展示了分区数据集，其中采用 VIP 驱动的本地排序，并在各机器间缓存远程特征。

连续排列；(b) 分区内的顶点按照其存储在 GPU 上的收益大小排序。每台机器都会将排序后顶点特征列表的一个前缀存储在 GPU 上；采用 VIP 排序时，这些顶点就是本地计算机访问最频繁的顶点。这种分区结构只需使用常量大小的额外内存，就能高效地判断顶点是远程还是本地的、确定本地顶点存储在哪台机器上，以及进行索引计算。图3 以图示形式展示了经过 VIP 排序的分区数据集。

4.2 远程顶点特征的复制

每台机器都会为远程分区中的顶点维护一份特征副本或静态缓存，其大小由给定的复制因子决定 α。为减少网络通信，每台机器缓存哪些顶点的特征，取决于这些顶点的 VIP 排名，详见第 3.2 节。

邻域采样代码准备好一个批次后，SALIENT++ 会将扩展邻域中的顶点划分为本地顶点和远程顶点。对于远程顶点子集，系统会通过哈希表进行高效查找，以确定本地计算机的缓存中是否有该远程顶点的特征。只有本地缓存中不存在的远程顶点才会被请求从远程机器获取。

4.3 小批次准备流水线

SALIENT++实现了一个深度多阶段流水线，使邻域采样、主机到设备的数据传输、机器之间的网络通信以及训练计算能够重叠进行。SALIENT++的流水线比 SALIENT（Kaler 等人，2022）所采用的流水线更为复杂，但仍保持了后者的易用性和效率。SALIENT++的流水线只需修改几行代码，即可替代图神经网络的标准数据加载器。

采样与切片SALIENT++ 使用经过修改的 SALIENT 优化版邻域采样与切片代码，通过共享内存

并行来准备小批次。原始的 SALIENT 批次准备代码将邻域采样和特征张量切片操作融合在一起。在 SALIENT++中，这些张量切片操作只针对采样邻域中存储在每台机器 CPU 内存中的顶点子集执行。对于采样邻域中的其余顶点，SALIENT++ 将其区分为以下几类：属于本地分区且其特征存储在 GPU 上的顶点；复制到本地计算机上的远程顶点；以及属于其他分区的远程顶点。

特征收集流水线 我们简要概述该流水线的操作。每个小批次的扩展邻域可能包含远程顶点、在 VIP 分析后其特征已缓存在本地计算机上的远程顶点，以及分散存储于 CPU 和 GPU 存储中的本地顶点。采样完成后，顶点会立即按其存储位置分类，以便正确检索顶点特征。与远程机器交互以协调远程特征检索的通信轮次，与主机到设备数据传输和设备到主机传输交错入队。分散存储于 CPU 和 GPU 存储中的副本特征和局部特征则在本地切片。最后，所有特征会拼接为一个张量，并重新排序，以便与邻域采样期间生成的消息流图兼容。

总共有 10 个小批次可以同时处于 SALIENT 的“在途”状态++ 流水线中，每个小批次由流水线中的一个独立阶段处理。流水线化使主机到设备数据传输与网络通信能够重叠，还能隐藏 CPU-GPU 数据传输和机器间通信相关的延迟。附录 中提供了对 SALIENT++的流水线逐阶段详细说明D。

5 评估

在本节中，我们评估 SALIENT 的性能++，并研究其与流水线化和 VIP 分析（如第3节所述）相关的优化措施所产生的影响。此外，我们还报告端到端性能结果和模型准确率，并结合现有分布式 GNN 训练系统的情况分析我们的性能表现。

5.1 实验设置

计算环境 我们的实验在一个由 16 台机器组成的集群上运行，每台机器均配备一颗 16 核（双路超线程）AMD CPU、128GB DRAM，以及一块拥有 24GB 显存的 NVIDIA A10G GPU。该集群使用 Amazon Web Services（AWS）ParallelCluster 软件进行配置，并采用 AWS *g5.8xlarge* 实例机群。此实例类型的网络 SLA 规定为

表 *2.* 数据集概要。边数表示经过标准预处理（例如，将图转换为无向图）后图中的边数。*mag240c* 数据集是异构图数据集*lsc-mag240* 中的论文间引文子图。

数据集	#顶点	#边	#特征	训练集 / 验证集 / 测试集
产品	2.4M	123M	100	197K / 39K / 2.2M
论文	111M	3.2B	128	1.2M / 125K / 214K
mag240c	121M	2.6B	768	110万 / 13.4万 / 8.8万

表 *3.* 第 5 节所有实验的图神经网络架构超参数。扇出用于训练。批次大小为每个 GPU 的批次大小。

数据集 图神经网络	#层数 隐藏维度		扇出	批次
产品 SAGE	3	256	(15, 10, 5)	1024
论文 SAGE	3	256	(15, 10, 5)	1024
mag240c SAGE	2	1024	(25, 15)	1024

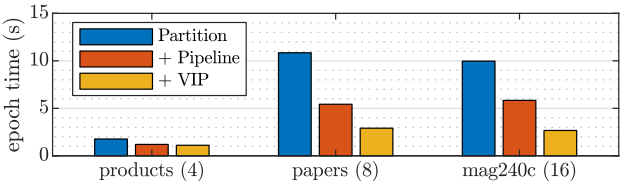


图*4*。 SALIENT 中流水线化和 VIP 优化的影响++。展示了通过逐步采用更多优化版本的 SALIENT 所获得的性能提升++对于 产 品（4 个分区）、论文 （8 个分区）和 *mag240c*（16 个分区）。每个基准测试所使用的 GNN 架构列于表 3 中。基于 VIP 的缓存的复制因子分别为 0。16、0。32 和 0。32 *240*，分别对应产品、论文和 *magc*。

25 Gbps。分布式通信通过 PyTorch 的 DistributedDataParallel 模块和 NCCL 后端实现。由于我们的集群由每台配备 1 块 GPU 的机器组成，因此使用 K 块 GPU 的实验涉及 K 台通过网络进行通信的独立机器。

数据集 我们在三个标准基准数据集上评估 SALIENT++ ：ogbn-products（产品）、ogbn- papers100M（论文）（Hu 等人，2020）、以及 lsc-mag240（*mag240c*）（Hu 等人，2021）。这些数据集中的图和训练集规模各不相同，其中 论文和 *mag240c* 是本研究开展时规模最大的两个开放基准数据集。详细信息请参见表 2 。所有图均被转换为无向图（如果其原本不是无向图），这是常见做法。对于 *mag240c*，我们在该图的同质论文到论文组件上进行训练。

GNN 架构 我们使用标准的 GraphSAGE（++ Hamilton 等人，2017）架构，并结合每个数据集最常用的超参数，评估了 SALIENT。表 3 列出了各个数据集所使用的 GNN 架构和超参数。

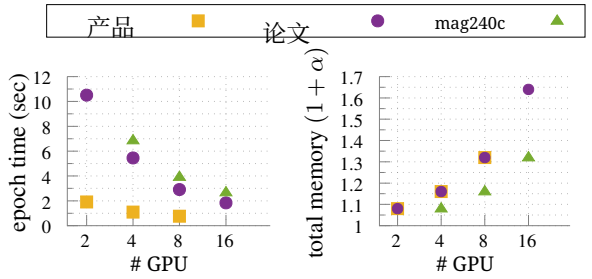


图 5。products、papers 和 mag240c 在所有机器上的可扩展性及总内存使用量。总内存以未复制数据集大小的倍数表示 $1+\alpha$ 其中 α 是 K-GPU 执行所采用的复制因子。

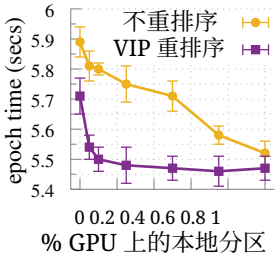


图 6。基于 VIP 的本地顶点排序对每轮次运行时间的影响。在 4 个 GPU 上使用论文数据集进行实验，复制因子为 $\alpha = 0.15$。报告的数值为 10 个轮次的平均值，竖线表示标准差。

5.2 SALIENT++ 性能分析

我们通过考察 SALIENT++ 各项组件优化的影响，并测量其在三个大图数据集上的端到端性能和可扩展性，来分析 SALIENT 的性能。

性能提升总结 图 4 总结了 SALIENT++的优化如何使其在分布式环境下，在 *prod-ucts*、论文和 *mag240c* 上的端到端训练性能逐步提升。无论是 论文还是 *mag240c* 基准测试，都观察到了显著的性能提升。相对而言，论文基准测试从流水线化和基于 VIP 的缓存中获益相当，而*mag240c* 基准测试则从流水线化基础上的缓存中略微获益更多。这主要是因为 *mag240c*的特征维度更大，因此与 论文相比，远程特征数据的通信相对更受吞吐量限制。

可扩展性 图 5（左图）展示了使用 SALIENT++在 2–16 个 GPU 上运行 products、papers 和 mag240c 基准测试时，每轮运行时间的缩短情况。对于这三项基准测试，报告的运行时间均包含每个训练轮次开始时填充流水线所需的预热时间；当每轮运行时间低于一秒后，可扩展性因此有所下降。products 基准测试从 2 个 GPU 增至 4 个 GPU 时，加速比达到 1.7×，但进一步扩展并未带来显著收益。papers 基准测试从 2 个 GPU 增至 4 个 GPU 时，加速比达到 1.9×，从 4 个增至 8 个 GPU 时又提升了 1.9×。mag240c 基准测试在 GPU 数量

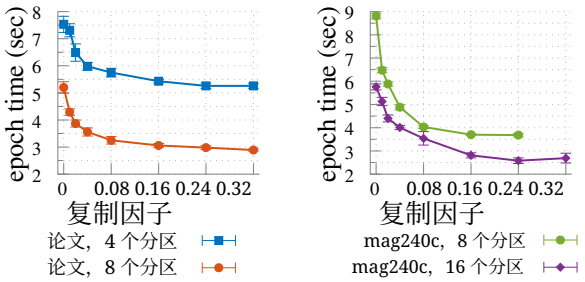


图 7。复制因子对每轮运行时间的影响。结果展示了 论文（左图）在 4 个和 8 个分区下，以及 *mag240c*在 8 个和 16 个分区下的表现；复制因子 α 在 0 到 0.32 之间变化。本次实验中，存储在 GPU 上的本地分区比例为：论文 为 90%，*mag240c*为 10%。报告的数据为 10 轮的平均值，竖线表示各轮运行时间的标准差。

从 4 增至 8 个 GPU 时，加速比达到 1.45× 加速比达到从 8 增至 16 个 GPU。SALIENT++的性能与先前采用完全复制进行分布式图神经网络训练的快速系统相当（Kaler 等人，2022），而 SALIENT++ 使用的内存则少得多。图5 （右图）显示了所有机器上用于存储顶点特征数据（包括复制的顶点）的总内存。请注意，使用K 台机器进行完全复制对应的复制因子为 $\alpha = K - 1$。

CPU 与 GPU 上的本地分区存储 图 6展示了运行论文基准测试时，提高存储在 GPU 上的本地分区比例对性能的影响。给定本地分区中顶点的排列顺序，以及要存储在 GPU 上的本地数据比例 $\beta\%$，SALIENT++ 会将前 $\beta\%$ 的顶点特征存储在 GPU 上。标记为“无重排序”的结果显示，按顶点的初始排列顺序对本地顶点排序时，每轮运行时间会随着 $\beta\%$ 的变化而近似线性下降。另一方面，标记为“VIP 重排序”的结果表明，按 VIP 值对本地顶点排序时，只需将本地分区数据的10 %存储在 GPU 上，便可有效消除数据传输瓶颈。

复制因子的影响 图 7 展示了在 论文 数据集上使用 4 和 8 个分区（左图），以及在 *mag240c* 数据集上使用 8 和 16 个分区（右图）时，增加复制的顶点特征数量对每轮运行时间的影响。图 7表明，在 0.08–0.16 和 0.16–0.32 的适中复制因子，分别足以在 4 和 8 个分区上最大限度地缩短每轮运行时间。论文 数据集。

SALIENT 的性能分解++ 图 8 展示了 SALIENT 中的性能瓶颈++ 在通过 VIP 分析引入流水线化和缓存前后的情况。[2] “启用流水线”和“启用流水线

[2]图 8 中的性能分解实验将所有本地节点特征存储在 GPU 上。因此，“流水线

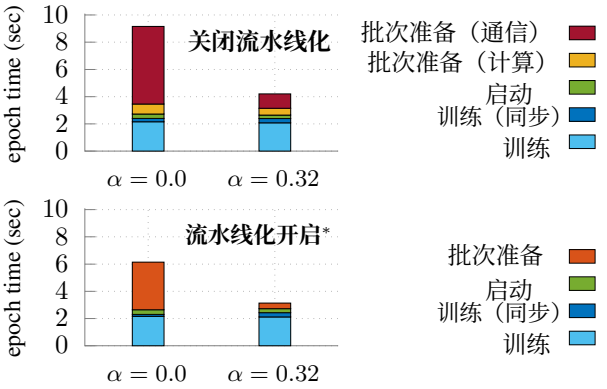


图 *8*。在 GPU 上存放所有本地节点特征时，SALIENT++ （增加了同步）在 8-GPU 执行 论文 任务中的性能分解。“关闭流水线化”分解最能准确隔离执行各项操作所花费的时间。“开启流水线化”分解仅增加了少量同步，以便将分布式批次准备与训练计算区分开来。批次准备（通信）和批次准备（计算）分别表示分布式批次准备中的通信和计算所花费的时间。在“开启流水线化”分解中，这两类合并为批次准备。启动表示准备第一个批次和/或填充流水线所花费的时间。训练（同步）表示并行工作线程在训练反向传播中的第一个集合通信操作处进行同步所花费的时间。训练表示训练所需的 GPU 计算时间。

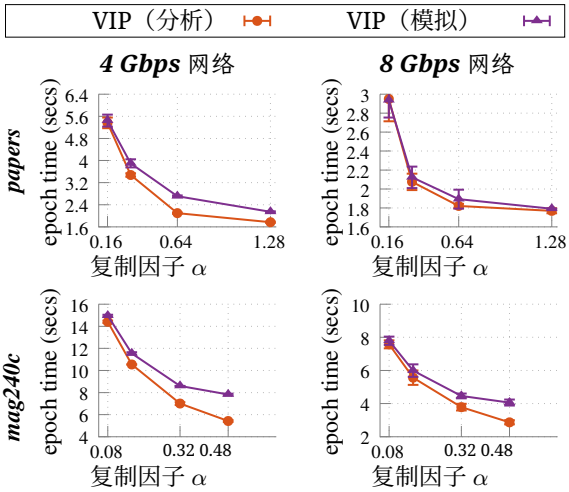


图*9*。比较慢速网络下 VIP-analytic 与 VIP-simulation 的端到端运行时间。展示了采用 VIP-analytic 和 VIP-simulation 缓存策略、使用不同复制因子时，*papers* 和*mag240c* 上的 16 节点执行每轮运行时间。

关闭”时的分解结果表明，对于 $\alpha = 0$ 而言，网络通信是分布式图神经网络训练的主要瓶颈；即使将通信进行流水线化以便与计算重叠，它仍然是瓶颈。使用 $\alpha = 0.32$ 进行缓存时，执行通信所耗费的时间足够短，因此几乎可以与程序中的其他计算完美重叠。

慢速网络上的性能图9展示了 SALIENT++在慢速网络上的性能，在这种情况下需要更高的复制因子来缓解通信瓶颈。我们使用 Linux 流量控制子系统中的令牌桶过滤器（TBF）排队规则来限制网络速度（Hubert 等人，2002）。在此设置中，我们比较第3节中表现最好的两种缓存策略：VIP（解析式）和 VIP（模拟式）。回顾第3节可知，这两种策略在复制因子较低时的性能往往非常相似，但随着复制因子增大，两者的表现会出现差异。在论文数据集上，这两种策略之间的相对差距逐渐扩大，最高达到30%，直到 $\alpha = 0.64$随后，当通信不再成为训练时间的瓶颈时，差距开始缩小。对于*mag240c* 数据集，其使用的节点特征比 6× 大论文，两者之间的相对 VIP（解析式）与 VIP（模拟式）之间的差距更大（在 $\alpha = 0.48$ 时最高可达 45%），且持续存在。是否优先选择 VIP（解析式）而非 VIP（模拟式），取决于工作负载特征（例如扇出和节点特征的大小）以及网络带宽预算。

5.3 SALIENT++性能的背景

为了结束实证评估，我们通过报告模型准确率结果、讨论预处理开销，并将我们的性能与现有系统进行比较，来说明 SALIENT++ 的性能。

模型准确率 SALIENT++中的优化不会影响模型准确率。SALIENT++ 与 PyTorch Geometric (PyG) 集成，并可使用通过 PyG 实现的现有图神经网络模型。无论如何，我们报告了产品、论文 和*mag240c* 的准确率，以便读者能够结合这些数据理解性能指标。为计算准确率，我们在 8 台机器上运行 SALIENT++ ，训练 30 轮，学习率固定为 0.001，每台机器的批次大小为 1024。[3] 推理过程中使用采样，论文 和产品的扇出数为 (20,20,20)，*mag240c*的扇出数为 (25,15)。在产品数据集上，3 层 SAGE架构的测试准确率为 0.785。在论文数据集上，3 层 SAGE架构的测试准确率为 0.646。在*mag240c*数据集上，2 层 GraphSAGE 架构的准确率为 0.651。[4]

关闭”系统 $\alpha = 0$在图 8 中略快于表 1 中的情况，后者的所有本地节点特征均位于 CPU 上。启用流水线的系统使用了SALIENT++的一个带有额外同步机制的流水线版本，因此速度略有下降，以便于在多个网络操作以及不同 CUDA 流上运行的并发计算之间分配时间。

[3]这些超参数并未针对准确率进行调优，但已知是合理的取值。[4]报告的是验证准确率，而非“test-dev”或“test-challenge”准确率；当时我们并未测量后两者

表 4. SALIENT++、DistDGL（公开代码）和 DistDGLv2（Zheng 等人，2022）的比较。所有数字均针对在3层 GraphSAGE架构（扇出数为 15、10、5）和256 个隐藏节点上运行的论文数据集报告。

系统	时间（秒）	注释
SALIENT++	2.9	8 个 NVIDIA A10G GPU，25 Gbps 网络 吞吐量，19.5 美元/小时。
DistDGL	37.0	与 SALIENT++ 使用相同的硬件。例- 来自 GitHub[5] 的示例分布式代码
DistDGLv2	≈ 5	64 块英伟达 T4 GPU，100 Gbps 网络 吞吐量，每小时 62.6 美元。[6]

预处理开销 SALIENT++ 的预处理开销分为两类：(a) 每次在数据集上执行之前产生的运行时开销；以及 (b) 仅产生一次、可在多次实验中摊销的数据集准备开销。SALIENT++ 在 papers 数据集上进行 8 节点执行、复制因子为 $\alpha = 0.32$ 时的运行时开销如下。从磁盘加载数据集大约需要 10 秒。对于扇出数 (15, 10, 5)，使用带稀疏转移权重矩阵和稠密的逐跳 VIP-向量的 PyTorch 实现计算 VIP 权重需要 11.8 秒。对于大型图，VIP 计算代码将矩阵行分批流式传输到 GPU，并重叠通信与数据传输。远程特征向量的通信以及相关的张量切片操作大约需要 22 秒。数据集预处理开销高度取决于所采用的图划分工作流。我们的（未经优化的）图划分工作流使用串行 METIS 对图进行分区，此外还使用内存有限的机器，因此必须使用交换文件。在此设置下，对 papers 数据集进行分区需要$\sim$2 小时，从该分区创建重排序数据集需要 30 分钟。经过优化的分区工作流可以显著更快地生成分区。例如，DistDGLv2（Zheng 等人，2022）报告称，使用 ParMETIS（Karypis 等人，1998）和 4 台大型多核机器进行分区时，运行时间为 12 分钟。SALIENT++ 不依赖于图划分的来源，并且可以与更具可扩展性的图划分代码结合使用。

与 DistDGL 的比较 我们比较了 SALIENT++ 与 DistDGL 在 *papers* 数据集上采用 Graph-SAGE 架构时的性能。表 4总结了这一比较结果。DistDGL 代码取自 DGL 的公开 GitHub 代码仓库，并在与评估 SALIENT++时相同的计算环境中运行。DistDGL 的公开分布式多 GPU 训练代码比 SALIENT 慢约 12。7× 倍++，在 8 块 GPU 上运行（每台机器配备 1 块 GPU）。当然，DGL 仍在积极开发中，其对高效图神经网络训练（尤其是在分布式环境中）的支持也在快速演进。因此，我们还报告了近期研究“DistDGLv2”（Zheng 等人，2022）中的性能结果。该研究介绍了 DistDGL 系统中用于提升分布式图神经网络训练性能的创新方法。虽然这些创新并非全部公开可用，但其报告的性能可用于对比我们使用 SALIENT++所取得的性能。对 DistDGLv2 和 SALIENT++ 所用硬件的全面比较++表明，SALIENT++的8块 GPU 运行时间约为 1.7×，在 *papers* 数据集上使用 64块 GPU 执行一个训练轮次时，比 DistDGLv2 报告的数值更快。SALIENT 始终能达到与 DistDGLv2 报告结果相近（通常更优）的性能，同时使用的资源显著更少。SALIENT++的8块 GPU 运行时间约为 1.7×，在 *papers* 数据集上使用 64块 GPU 执行一个训练轮次时，比 DistDGLv2 报告的数值更快。实现更短的每轮耗时所需的 GPU 更少、8× 网络带宽更小，4×硬件成本也更低。

6 结论与未来工作

我们提出了 SALIENT++，一个用于大规模图上图神经网络训练与推理的分布式多 GPU 系统。该系统构建于 SALIENT 之上；作为此前最先进的系统，SALIENT 通过快速采样和流水线化实现高效率与可扩展性，但代价是需要在所有机器上完整复制数据。在 SALIENT++ 中，我们分布式存储顶点特征数据，并通过分析机外（即远程）顶点的访问模式，提出一种缓存策略，复制少量访问频率最高的顶点特征，以解决由此产生的通信瓶颈。通过对从通信到计算的所有操作进行深度流水线处理，SALIENT++ 在仅消耗 SALIENT 所需存储空间一小部分的情况下，保留了 SALIENT 的高效率与可扩展性。

未来的一个研究方向是进一步将访问模式分析应用于改进初始图划分，以期在不依赖缓存的情况下减少通信量。这需要在图划分目标中纳入顶点纳入概率，并兼顾边割和负载均衡。此外，分层图划分或许能比机器间通信更好地利用 GPU 之间更高的机内带宽。另一个未来研究方向是探索如何在多台机器之间分布图结构。分布式存储图会给节点特征的多轮通信带来不小的挑战，但解决这一问题将进一步降低内存消耗，并有望处理规模比当前最大基准测试大几个数量级的图。

投稿。OGB 排行榜上公布的先前结果表明，对于该数据集上的 GraphSAGE，此结果具有一致性。[6]硬件详情来自 DistDGLv2 论文（Zheng 等人，2022），定价信息则来自亚马逊云科技，针对所报告使用的*g4dn.metal* 实例类型。

致谢

本研究在 SuperCloud 计算集群 https://supercloud.mit.edu 和 Satori 计算集群 https://mit-satori.github.io上完成。本研究由 MIT-IBM Watson人工智能实验室资助，部分由美国空军研究实验室和美国空军人工智能加速器资助，并依据合作协议编号 FA8750-19-2-1000 开展。本文所述观点和结论均为作者观点，不应被解读为美国空军或美国政府明示或暗示的官方政策。尽管本文包含任何版权声明，美国政府仍有权为政府用途复制和分发本文的重印本。

参考文献

Abadi, M., Agarwal, A., Barham, P., Brevdo,E., Chen, Z.,Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin,M.,Ghemawat, S., Goodfellow, I., Harp, A., Irving, G.,Isard,M., Jia,Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Lev-enberg, J., Man´e,D.,Monga, R., Moore,S., Murray, D.,Olah, C., Schuster, M., Shlens, J., Steiner, B.,Sutskever,I., Talwar,K., Tucker, P., Vanhoucke, V., Vasudevan,V.,Vi´egas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow：异构系统上的大规模机器学习，2015 年。https://www.tensorflow.org/。Bottou, L., Curtis, F. E., and Nocedal, J. 大规模机器学习的优化方法。*SIAM Rev.*，60 (2):223–311，2018 年。Cai, Z., Yan, X., Wu, Y., Ma, K., Cheng, J., and Yu, F. Dgcl：用于分布式图神经网络训练的高效通信库。载于第十六届欧洲计算机系统会议论文集，EuroSys '21，第 130–144 页，纽约，纽约州，美国，2021 年。计算机协会。国际标准书号 9781450383349。数字对象标识符：10.1145/3447786.3456233。网址 https://doi.org/10.1145/3447786.3456233。Chen, J. and Luss, R. 使用有偏但一致的梯度估计器进行随机梯度下降。预印本 arXiv:1807.11880，2018 年。Chen, J., Ma, T., and Xiao, C. FastGCN：通过重要性采样实现图卷积网络的快速学习。载于*ICLR*，2018 年。Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-GCN：用于训练深层、大规模图卷积网络的高效算法。载于*KDD*，2019 年。Dong, J., Zheng, D., Yang, L. F., and Karypis, G. 面向巨型图 CPU-GPU 混合训练的全局邻居采样。发表于*KDD*，2021年。Gandhi, S. and Iyer, A. P. P3：大规模分布式深度图学习。发表于*OSDI*，2021年。Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. 面向量子化学的神经消息传递。发表于*ICML*，2017年。Hamilton, W. L., Ying, R., and Leskovec, J. 大型图上的归纳式表示学习。发表于*NIPS*，2017年。Hoang, L., Chen, X., Lee, H., Dathathri, R., Gill, G., and Pingali, K. 面向大型图深度学习的高效分布式处理。发表于*GNNSys*，2021年。Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. 开放图基准：面向图机器学习的数据集。预印本 arXiv:2005.00687，2020年。Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. OGB-LSC：面向图机器学习的大规模挑战。*arXiv* 预印本 *arXiv:2103.09430*，2021年。Hubert, B. 等。Linux 高级路由与流量控制指南。*Netherlabs BV*，1:99–107，2002年。Jia, Z., Lin, S., Gao, M., Zaharia, M., and Aiken, A. 借助 Roc 提高图神经网络的准确性、可扩展性和性能。发表于*MLSys*，2020年。Kaler, T., Stathas, N., Ouyang, A., Iliopoulos, A.-S., Schardl, T., Leiserson, C. E., and Chen, J. 通过快速采样和流水线化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，4:172–189，2022年。Karypis, G. and Kumar, V. Metis：用于划分非结构化图、划分网格以及计算稀疏矩阵填充缩减排序的软件包。1997年。Karypis, G., Schloegel, K., and Kumar, V. ParMETIS——并行图划分与稀疏矩阵排序库，2.0版。明尼苏达大学，明尼阿波利斯，明尼苏达州，10，1998年。Kipf, T. N. and Welling, M. 使用图卷积网络进行半监督分类。发表于*ICLR*，2017年。Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. 门控图序列神经网络。发表于*ICLR*，2016年。Li, Y., Yu, R., Shahabi, C., and Liu, Y. 扩散卷积循环神经网络：数据驱动的交通预测。发表于*ICLR*，2018年。

Lin, Z., Li, C., Miao, Y., Liu, Y., and Xu, Y. Pagraph：通过计算感知缓存扩展大型图上的图神经网络训练。发表于第*11*届 *ACM* 云计算研讨会论文集，SoCC '20，第401–415页，纽约，纽约州，美国，2020年。计算机协会。国际标准书号9781450381376。数字对象标识符：10.1145/3419111.3421281。网址：https://doi.org/10.1145/3419111.3421281。

Liu, Z., Wu, Z., Zhang, Z., Zhou, J., Yang, S., Song, L., and Qi, Y. 用老虎机采样器训练图神经网络。载于神经信息处理系统进展，第 33 卷，第 6878–6888 页，2020。

Ma, L., Yang, Z., Miao, Y., Xue, J., Wu, M., Zhou, L., and Dai, Y. NeuGraph：在大型图上进行并行深度神经网络计算。载于*USENIX ATC*，2019。

Md, V., Misra, S., Ma, G., Mohanty, R., Georganas, E., Heinecke, A., Kalamkar, D., Ahmed, N. K., and Avancha, S. DistGNN：面向大规模图神经网络的可扩展分布式训练。载于高性能计算、网络、存储与分析国际会议论文集，第 1–14 页，2021。

Min, S. W., Wu, K., Huang, S., Hidayeto˘glu, M., Xiong, J., Ebrahimi, E., Chen, D., and mei Hwu, W. 采用面向GPU的数据通信架构进行大型图卷积网络训练。预印本 arXiv:2103.03330，2021。

Mohoney, J., Waleffe, R., Xu, H., Rekatsinas, T., and Venkataraman, S. Marius：在单台机器上学习大规模图嵌入。载于第*15*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 21*），第 533–549 页。USENIX协会，2021年7月。国际标准书号 978-1-939133-22-9。网址：https://www.usenix.org/conference/osdi21/presentation/mohoney。

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch：一种命令式风格、高性能的深度学习库。载于*NIPS*，2019年。

Ramezani, M., Cong, W., Mahdavi, M., Sivasubra maniam, A., and Kandemir, M. GCN 与 GPU 相遇：将“何时采样”与“如何采样”解耦。载于 *NeurIPS*，2020。

Veli˘ckovi´c, P., Cucurull, G., Casanova, A., Romero, A., Li`o, P. 和 Bengio, Y. 图注意力网络。发表于 *ICLR*，2018 年。

Waleffe, R., Mohoney, J., Rekatsinas, T. 和 Venkatara- man, S. Marius++：在单台机器上大规模训练图神经网络。*arXiv* 预印本 *arXiv:2202.02365*，2022。Wan, C., Li, Y., Li, A., Kim, N. S. 和 Lin, Y. Bns-gcn：通过分区并行和随机边界节点采样，高效地对图卷积网络进行全图训练。载于 Marculescu, D., Chi, Y. 和 Wu, C.（编），机器学习与系统论文集，第 4 卷，第 673–693 页，2022。网址：https://proceedings.mlsys.org/paper/2022/file/d1fe173d08e959397adf34b1d77e88d7-Paper.pdf。

Wang, L., Yin, Q., Tian, C., Yang, J., Chen, R., Yu, W., Yao, Z., and Zhou, J. FlexGraph：用于图神经网络训练的灵活高效分布式框架。载于 *EuroSys*，2021a。

Wang, Y., Feng, B., Li, G., Li, S., Deng, L., Xie, Y., and Ding, Y. GNNAdvisor：面向 GPU 上图神经网络加速的自适应高效运行时系统。载于 *OSDI*，2021b。

Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., and Leiserson, C. E. 比特币反洗钱：利用图卷积网络开展金融取证实验。载于 *KDD* 金融异常检测研讨会，2019。

Wu, Y.、Ma, K.、Cai, Z.、Jin, T.、Li, B.、Zheng, C.、Cheng, J. 和 Yu, F. Seastar：面向图神经网络的以顶点为中心的编程。在*EuroSys*，2021 年。

Xu, K.、Hu, W.、Leskovec, J. 和 Jegelka, S. 图神经网络的能力有多强？在*ICLR*，2019 年。

Yang, J.、Tang, D.、Song, X.、Wang, L.、Yin, Q.、Chen, R.、Yu, W. 和 Zhou, J. GNNLab：一种面向 GPU 上基于采样的图神经网络训练的分解式系统。在 *EuroSys*，第 417–434 页，2022 年。

Ying, R.、He, R.、Chen, K.、Eksombatchai, P.、Hamilton, W. L. 和 Leskovec, J. 面向网络规模推荐系统的图卷积神经网络。在*KDD*，2018 年。

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT：基于图采样的归纳学习方法。发表于*ICLR*，2020。

Zheng, D., Ma, C., Wang, M., Zhou, J., Su, Q., Song, X., Gan, Q., Zhang, Z., and Karypis, G. DistDGL：面向十亿级图的分布式图神经网络训练。发表于*IA3*，2020。

Zheng, D.、Song, X.、Yang, C.、LaSalle, D. 和 Karypis, G. 的分布式 CPU 与 GPU 混合训练，用于

在十亿级图上训练图神经网络。预印本 arXiv:2112.15345，2021 年。

Zheng, D., Song, X., Yang, C., LaSalle, D., 和 Karypis, G. 面向十亿级异构图的图神经网络分布式 CPU 与 GPU 混合训练。载于 第 *28* 届 *ACM SIGKDD*知识发现与数据挖掘大会论文集，KDD '22，第 4582–4591 页，纽约，纽约州，美国，2022 年。计算机协会。国际标准书号 9781450393850。数字对象标识符：10.1145/3534678.3539177。网址 https://doi.org/10.1145/3534678.3539177。

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. 面向深度大规模图卷积网络训练的层依赖重要性采样。发表于*NeurIPS*，2019 年。

A 附录：工件娱乐下载娱乐主管

A.1 摘要

本节介绍与本文相关的软件制品。代码通过 GitHub 发布，地址为 https://github.com/MITIBMxGraph/SALIENT_plusplus_制品，可用于执行本文所述的实验。我们提供了以下脚本：(a) 运行模拟实验，以比较不同的缓存策略并生成图2的数据；(b) 运行分布式实验，生成表1 和图4–7的数据。有关运行这些脚本的详细说明，请参阅专为制品评估编写的 README 文件，地址为 https://github.com/MITIBMxGraph/SALIENT_plusplus –artifact/blob/main/自述文件 .md.

A.2 制品检查清单（元信息）

• **算法：** SALIENT++ 用于图神经网络的分布式训练算法，以及用于图神经网络缓存策略的 VIP 分析算法。

• **程序：** PyTorch、CUDA

• **编译：** gcc/g++ 版本 7 或更高；nvcc 版本 11。

• **数据集：** OGB 的节点分类基准数据集。

• **运行环境：** Ubuntu 20.04（或较新的 Linux 发行版），并安装英伟达驱动程序。

• **硬件：** 具备充足显存的英伟达 GPU。分布式实验需要配备 GPU 节点的 SLURM 集群。模拟实验需要最多 200 GB 的主存。

• **实验：** 本地模拟实验，用于比较不同缓存策略在远程顶点通信量方面的表现；分布式实验，用于分析不同优化、不同复制因子以及 GPU 内存中存储的数据百分比对每轮训练时间的影响。

•**大约需要多少磁盘空间？：** 所有实验共需 1.5 TB，若仅运行大量子集实验，则需 100 GB。

• **准备工作流程大约需要多长时间？：** 具备相关经验并可使用硬件/集群的情况下，需要 1–2 小时。

• **完成实验大约需要多长时间？：** 如果使用预处理数据集，所有实验需要 4–12 小时。

• **是否公开可用？：** 是

• **代码许可证：** Apache 2.0 许可证

• **数据许可证：** 亚马逊许可证和 ODC-BY。

•**档案数字对象标识符：**
https://doi.org/10.5281/zenodo.7889203

A.3 描述

A.3.1 获取方式

代码可在 GitHub 上获取，网址为 https://github.com/MITIBMxGraph/SALIENT_plusplus_工件。代码仓库中提供了用于简化运行该工件流程的脚本，位于实验目录中。有关如何运行这些脚本的说明，请参阅代码仓库顶层的 README 文档，网址为 https://github.com/MITIBMxGraph/SALIENT_plusplus_artifact/blob/main/README.md。

A.3.2 硬件依赖项

运行该软件工件的最低要求如下。

使用单台机器（配备或不配备 GPU）即可在 ogbn-papers100M 数据集上进行模拟实验，该机器的主存至少为 160GB。对于内存容量较低的机器，可以使用交换文件来运行这些实验，最好将交换文件放在高速 SSD 上。较小的 ogbn-products 数据集上的模拟实验所需内存不足 10 GB。

分布式多 GPU 实验需要访问具有 GPU 节点的 SLURM 集群，或配备多个 GPU 的单台机器。选择使用配备多个 GPU 的单台机器的用户，应向实验脚本传递适当的标志，以表明他们正在本地运行脚本。可以通过云服务及配套软件包获取具有 GPU 节点的 SLURM 集群访问权限。例如，在亚马逊云科技上，可以使用 ParallelCluster 软件启动 SLURM 集群。

根据所使用的硬件和可用磁盘空间，某些实验可能无法进行。我们已尽力降低在最大数据集上运行性能实验所需的内存，并预计配备 16GB 内存的 GPU 和 128GB 主存的机器足以运行全部或几乎全部分布式实验。

A.3.3 软件依赖项

机器上必须安装相对较新的英伟达驱动程序。对于分布式实验，需要一个 SLURM

集群。其余外部软件依赖项可使用 conda 包管理器解决。

A.3.4 数据集

用于节点属性预测的图数据集取自 Open Graph Benchmark (OGB) 代码仓库。为了减少实验所需的时间和最低硬件资源，我们提供了下载预处理后图数据的选项。如果选择不下载预处理后的图，那么在新图上首次执行代码时，系统会从 OGB 下载该图并在本地进行预处理。请注意，对于分布式实验，还需要进行额外的预处理，以生成分区标签并按分区重新排列顶点；可使用提供的脚本完成此操作，具体说明见制品 README。

A.4 安装

我们建议参阅 https://github.com/MITIBMxGraph/SALIENT_plusplus_artifact/blob/main/INSTALL.md。以下是安装流程概要。

在 Python 环境中安装：我们提供了在 Python 虚拟环境中安装该制品的说明。此安装方法既适用于单机实验，也适用于分布式多 GPU 实验。下面总结了在 conda 环境中安装该制品的说明。

```
# 下载并安装 miniconda wget
https://repo.anaconda.com/miniconda/\ Miniconda3-py38_4.
10.3-Linux-x86_64.sh bash Miniconda3-py38_4.10.3-Linux-x86_64.
sh

# 为该制品创建 conda 环境 conda create -n salientplus
python=3.9.5 -y conda activate salientplus

# 安装 PyTorch、PyG、OGB、prettytable conda install -y pytorch==1.13.1
torchvision==0.14.1 \ torchaudio==0.13.1 pytorch-cuda=11.7 -c pytorch -c
nvidia conda install -y -c conda-forge ogb conda install -y pyg -c pyg -c
conda-forge conda install -y pytorch-sparse -c pyg conda install -y -c
conda-forge nvtx conda install -y -c conda-forge matplotlib conda install -y -
c conda-forge prettytable

# 安装 fast_采样器 cd fast_采
样器 python setup.py install
cd ..

#（可选）安装 METIS # - 已省略，请参阅代码仓库中的 INSTALL.md
```

A.5 实验工作流程

我们建议参阅 GitHub 代码仓库中的工件评估文档：https://github.com/MITIBMxGraph/SALIENT_plusplus_artifact/blob/main/README.md。我们在此概述实验工作流程。除非另有说明，所有文件路径均相对于代码仓库中的实验目录。

实验的初始设置

我们提供了两个实用脚本，以简化下载预处理数据集和分区的流程。配置脚本（configure_for_environment.py）会根据可用磁盘空间以及实验中要使用的 GPU（和分区）数量上限，确定要下载哪些数据集。另一个脚本（download_datasets_fast.py）会下载配置过程中选定的数据集。有关如何使用这些设置脚本，请参阅制品代码仓库 README。

模拟实验

此脚本`run_sim_experiments_paper.sh` 会在 ogbn-papers100M 数据集上运行模拟实验，以复现图 2中的结果。此脚本假定已使用上述设置脚本下载了 8 分区的 ogbn-papers100M 数据集。

用于运行自定义模拟实验的驱动脚本是 run_sim_experiments.sh。自定义模拟脚本的选项已记录在制品 README 中。

分布式多 *GPU* 实验

这些实验需要使用带有 GPU 节点的 SLURM 集群，或配备多个 GPU 的本地计算机。每个分布式实验脚本都提供用于自定义实验的选项，并会在实验完成后在终端中显示结果表。

所提供脚本的摘要如下：

- •实验_优化_`impact.py` 复现表1 和图4中的实验，以展示不同优化对每轮运行时间的端到端影响。
- • 实验_变化_复制_`factor.py`复现图5 和7 中的实验，以展示随着分布式节点数量增加时的性能可扩展性，以及每轮运行时间与复制因子之间的关系。
- •实验_变化_GPU_`percent.py` 复现图6，以分析在 GPU 上存储特征对性能的影响。
- • 实验_accuracy.py 对数据集执行端到端训练，并报告验证集和测试集上的准确率。

如果要在 SLURM 集群上运行这些脚本，则必须通过编辑文件顶部的 SLURM_CONFIG 变量来配置所提供的实验驱动脚本 exp_driver.py。更多详情请参阅代码仓库中的制品评估指南。如果要在配备多个 GPU 的单台机器上本地运行这些脚本，则必须向脚本传入命令行参数 --run_local 1。

希望运行自定义实验的用户可以直接使用核心实验驱动脚本 exp_driver.py。

A.6 评估与预期结果

完成模拟实验后，将生成一张表格，展示不同缓存策略下图神经网络训练的通信量。该表格复现了图2中的数据。

完成分布式 GPU 实验后，将生成多张表格，呈现表1以及图 4–7 的结果。参与者可根据硬件和时间限制，选择在不同的数据集上或使用不同的参数执行这些实验。

A.7 实验定制

可以进行以下实验定制。仿真和分布式实验脚本提供了命令行选项，可使用不同的数据集和参数运行。实验驱动脚本 exp_driver.py 可直接用于在多机多 GPU 环境下，使用不同的数据集、模型架构和训练参数开展定制性能实验。该软件也可独立于实验驱动脚本使用。随附的预处理实用脚本可用于对图进行分区，并根据图分区对数据集重新排序。

B 代码仓库

除了专注于基准测试和可复现性的制品仓库外，++面向通用用途的 SALIENT 实现可从以下地址获取：https://github.com/MITIBMxGraph/SALIENT_plusplus。

C P命题 1中逐层概率的证明

我们首先考虑某个顶点$u \in \mathcal{V}$ 被采样为某个小批次 $\mathcal{B}$ 的 1 跳邻居节点的概率。这等于对任意顶点 v，以下条件全部成立的概率： v 被纳入小批次， u 是 v的邻居节点，并且 u 在 v的直接邻居中被采样。若用 $\mathcal{N}_h^s(\mathcal{B})$ 表示扩展邻域中与 $\mathcal{B}$相距恰好 h 个扩展步数后被采样的顶点，则有：

$$
\begin{aligned}
p^{[1]}(u) &= \Pr[u \in \mathcal{N}_1^s(\mathcal{B})] \\
&= \Pr\left[\bigcup_{v \in \mathcal{N}_1^{\intercal}(u)} \Big((v \in \mathcal{B}) \cap (u \in \mathcal{N}_1^s(v))\Big)\right] \\
&= 1 - \Pr\left[\bigcap_{v \in \mathcal{N}_1^{\intercal}(u)} \Big((v \notin \mathcal{B}) \cup (u \notin \mathcal{N}_1^s(v))\Big)\right] \\
&= 1 - \prod_{v \in \mathcal{N}_1^{\intercal}(u)} \Pr[(v \notin \mathcal{B}) \cup (u \notin \mathcal{N}_1^s(v))] \\
&= 1 - \prod_{v \in \mathcal{N}_1^{\intercal}(u)} \Big(\Pr[v \notin \mathcal{B}] \\
&\qquad\qquad + \Pr[(u \notin \mathcal{N}^s(v)) \cap (v \in \mathcal{B})]\Big) \\
&= 1 - \prod_{v \in \mathcal{N}_1^{\intercal}(u)} \Big(1 - p^{[0]}(v) + \big(1 - t_1(u,v)\big)\, p^{[0]}(v)\Big) \\
&= 1 - \prod_{v \in \mathcal{N}_1^{\intercal}(u)} \Big(1 - t_1(u,v)\, p^{[0]}(v)\Big).
\end{aligned}
$$

第 4 行中的乘积由小批次中各顶点之间邻域采样独立性的假设推出。其中每个因子都是 u未被采样为 v的邻居节点的概率。对于有向图，当我们用 $\mathcal{N}_1^{\intercal}(u)$ 表示并集、交集和乘积的索引时，指的是 u的出向直接邻居。对于无向图， $\mathcal{N}_1^{\intercal}(u) = \mathcal{N}_1(u)$。

通过类似的推理，在扩展某个小批次的邻域时，顶点 u 恰好在 ℓ 跳数之外被采样的概率是

$$
\begin{aligned}
p^{[h]}(u) &= \Pr[u \in \mathcal{N}_h^s(\mathcal{B})] \\
&= \Pr\left[\bigcup_{v \in \mathcal{N}_1^{\intercal}(u)} \Big((v \in \mathcal{N}_{h-1}^s(\mathcal{B})) \cap (u \in \mathcal{N}_1^s(v))\Big)\right] \\
&= \cdots \\
&= 1 - \prod_{v \in \mathcal{N}_1^{\intercal}(u)} \Big(1 - t_h(u,v)\, p^{[h-1]}(v)\Big),
\end{aligned}
$$

其中省略的步骤与上述针对 $p^{[0]}(u)$ 的步骤类似。

D SALIENT++ 流水线阶段详解

小批次准备过程按以下阶段进行。阶段 1 获取下一个已处理的小批次by

SALIENT++ 来自邻域采样器。阶段 2 执行全对全通信，以广播每台机器将发送/接收的远程顶点数量。阶段 3 将阶段 2 的元数据传输到 CPU，以便分配大小合适的张量。阶段 4 执行全对全通信，每台机器i 向每台机器发送 j 一个节点列表，这些节点的特征位于机器 j 本地，且为机器 i所需。阶段 5 接收其他机器请求的节点列表，将其全局标识符映射为本地标识符，并执行设备到主机传输，使请求的节点列表可从 CPU 内存访问。阶段 6 启动一个异步工作线程，执行“掩码选择”操作，根据所引用的节点存储在本地 CPU 内存还是 GPU内存中，将每个请求的节点索引列表拆分为两组。阶段 6 启动的后台线程还会对特征存储在 CPU 内存中的请求节点执行张量切片。请注意，看似无害的“掩码选择”操作会将索引划分为 CPU 组和 GPU 组；如果在阶段 5 于 GPU 上执行该操作，可能会导致设备同步，因此我们将此任务交由后台CPU线程处理。阶段 7 启动主机到设备数据传输，将阶段 6 的结果发送到 GPU。阶段 8 在 GPU 上对本地存储于 GPU 上的请求节点特征执行张量切片，并合并 CPU/GPU 结果，使每台远程机器请求的特征都位于单个张量中。此外，阶段 8 对机器的本地特征缓存进行切片，以提取当前机器所需的远程特征 [7] 阶段 9 执行全对全通信，以传输所有请求的远程特征。阶段 10 将收到的远程特征合并为单个张量，并对其进行置换，使其与该批次本地生成的消息流图保持一致。

[7]这些操作在流水线中的位置在一定程度上是任意的，因为这些操作不依赖非局部信息。