# 通过设备拓扑感知图部署加速分布式深度神经网络训练

Shiwei Zhang、Xiaodong Yi、Lansong Diao、Chuan Wu、Siyu Wang 和 Wei Lin

**摘要**—本文介绍了 TAG，这是一种自动化系统，可为设备和拓扑异构的机器学习集群推导出优化的深度神经网络训练图，并将其部署到任意设备拓扑上，从而加快训练。我们创新性地将深度神经网络计算图和设备拓扑图共同作为图神经网络（GNN）的输入，并将 GNN 与基于搜索的方法相结合，以快速确定优化的分布式训练策略。为减少异构集群中的通信，我们进一步探索了一种无损梯度压缩技术，并求解一个组合优化问题，以自动应用该技术，从而最大限度地缩短训练时间。我们使用多种具有代表性的深度神经网络模型和设备拓扑对 TAG 进行了评估，结果表明，与现有方案相比，其训练加速比最高可达 4.56 倍。TAG 无需大量微调，即可为未见过的深度神经网络模型和设备拓扑生成高效的部署策略。

**索引词**—分布式系统、机器学习

arXiv:2302.06126v1 [cs.LG] 13 Feb 2023

## 1 引言

深度学习（DL）已在计算机视觉 [1], [2], 自然语言处理 [3], [4],推荐系统 [5],等多个领域得到了广泛应用。近期的深度神经网络（DNN）模型通常包含大量参数（例如 BERT [6] 拥有超过 3.4 亿个参数），以实现卓越性能 [3], [6]。如今，使用集群中的数十或数百个 GPU 进行大规模分布式训练，已成为训练此类模型的常态。

最先进的分布式训练主要利用同构集群，例如，使用 8 个 NVIDIA V100 GPU 训练 Bert [7]。然而，现代人工智能云通常托管着多种服务器类型，并配备不同的设备（例如 A100、V100 和 P100 GPU）。若每个训练任务只能分配相同类型的 GPU/机器，很可能会造成资源闲置且分散，例如一台机器上有 1 个 V100 GPU，另一台机器上有 2 个 P100 GPU。由于现有深度学习框架缺乏对异构集群训练的有效支持，这些分散资源往往无法分配给同一任务，因此大多处于闲置状态。要更好地利用昂贵的人工智能基础设施，关键在于实现异构设备上的高效分布式训练。

此外，人工智能云中设备之间的互连和带宽通常各不相同，这是由于链路类型不同（例如机器内部的 PCIe 或 NVLink）、共置级别不同（例如机器是否位于同一机架）等因素造成的。这进一步加剧了用于训练深度神经网络模型的机器学习（ML）集群的异构性。

为了尽可能加快训练，在异构且分散的资源上部署深度神经网络模型时，需要做出多个相互关联的决策：每个算子应放置在哪台机器的哪个设备上（操作放置 [8], [9]）？是否应在多个设备上复制一个或一组算子，以进行数据并行训练（操作复制 [10], [11]）？在复制的算子之间进行参数同步时，应使用 AllReduce [12], [13] 还是参数服务器（PS）架构 [14], [15]？所产生的梯度是否应由某些算子进行压缩，以减少设备间通信（梯度压缩 [11], [16]）？这些决策共同构成了一个呈指数级增长的策略空间。当前的实践往往退而采用启发式方法，每次只考虑策略空间的一个方面 [17], [18],，因而得到的解决方案效率较低，甚至不可行。

早期将深度神经网络模型部署到异构计算资源上的研究，采用强化学习和神经网络来寻找分布式训练策略 [10], [18], [19]。然而，这些模型无法泛化到不同的设备拓扑，并且每遇到一种新的资源配置，都需要从头开始训练。因此，它们并不适用于人工智能云，因为每个作业都会采用新的资源配置。如何针对未见过的设备拓扑快速找到分布式训练策略，仍有待探索。

我们提出了 TAG[1]，这是一种自动化的深度神经网络部署框架，可针对给定的深度神经网络模型及异构资源，高效生成优化的分布式训练策略。TAG 将异构图神经网络（GNN） [20] 与基于搜索的方法相结合，对操作复制、放置、参数同步和梯度压缩进行细粒度决策。

TAG 的主要贡献总结如下：▷ 提出了一种自动化的 DNN 部署框架，可针对给定的设备集合和设备间拓扑生成带有操作级复制的优化训练图，从而加速训练。该框架会自动插入必要的操作，以确保修改前后的原始 DNN 计算图在数学上保持等价。

▷ 设计了一种异构图神经网络，以计算图和设备拓扑作为输入，并学习一种可泛化的策略，引导蒙特卡洛树搜索（MCTS） [21], [22] 制定高效的操作放置方案。

▷ 为了进一步降低通信开销，我们采用充分因子广播（SFB） [23], 这一无损梯度

1. 我们计划将 TAG 开源。

压缩技术，并构建图割问题，以自动确定训练图中的子图复制和 SFB 的应用。

▷ 我们在具有代表性的深度神经网络模型和各种设备拓扑上进行了大量实验。与使用 NCCL AllReduce 的数据并行方法以及异构集群中的最先进训练方案相比，TAG 的加速比最高可达 4.56 倍。它无需大量微调，即可为未见过的深度神经网络模型和未见过的设备拓扑生成高效的部署策略。

## 2 背景

### 2.1 分布式深度神经网络训练

在 TensorFlow [24]、PyTorch [25], 和 MXNet [26], 等现代机器学习框架中实现的深度神经网络模型，可以表示为有向无环图（DAG），称为相应深度神经网络模型的计算图。在计算图中，每个节点都是一个操作（算子），连接各算子的边表示张量。算子会被放置在计算设备（例如 GPU）上执行。如果生成张量的算子与使用该张量的算子被放置在不同的设备上，就需要在设备之间传输该张量。

深度神经网络训练是一个迭代过程。在每次迭代中，系统会对一批训练数据执行前向计算，随后通过反向传播计算梯度并更新模型参数。数据并行（DP）和模型并行（MP）是分布式训练的两种主要范式 [6], [27]。在经典数据并行中，每个设备都保存一份完整的模型副本，并独立处理一批数据。模型并行则将深度神经网络模型的不同部分放置在不同设备上，训练过程中会在设备之间传递中间张量。

许多研究都探索了深度神经网络训练中的设备放置、算子复制以及混合数据并行/模型并行。GDP [9]、GO [28], 、PlaceTo [8] 和 HeteroG [10] 使用图神经网络提取计算图信息，并生成算子的设备分配方案。HDP [18] 和 Spotlight [29] 使用强化学习（RL）训练长短期记忆网络，以做出放置决策。FlexFlow [30] 使用马尔可夫链蒙特卡罗（MCMC）搜索算法搜索算子放置方案，从而实现混合并行。REGAL [31] 使用强化学习和遗传算法（GA）共同优化放置和调度。Pesto [32]将放置和调度建模为整数线性规划（ILP），并使用现成的整数线性规划求解器求解。现有基于学习的系统都不支持未见过的设备拓扑；设备拓扑发生变化时，必须重新训练相应的图神经网络或长短期记忆网络模型。

### 2.2 共享人工智能基础设施

先进的人工智能云通常包含多个服务器机架，配备少数几种具有代表性的不同代际 GPU。生产环境中的 AI 云通常会将同一类型的 GPU 分配给同一个训练任务，并尽可能将这些 GPU 安排在同一台机器上，或彼此相邻的机器上 [33]。这种资源分配往往会造成资源碎片化，例如，少量未分配的 GPU（1 至 2 个）分散在不同机器上，导致昂贵的人工智能资源大量浪费。此外，申请大量同一型号 GPU 的训练任务还会面临漫长的排队时间 [34], ，因为它们必须等待所有资源都可用。

现有的深度学习框架（TensorFlow、PyTorch、MXNet 等）和训练策略大多支持在同构集群上进行高效的分布式训练，但在异构资源（即计算能力和内存容量各不相同的 GPU，以及变化不一的设备间带宽）上，训练效率会显著下降。如何高效利用分散的异构资源，是一个备受关注的设计问题。

### 2.3 充分因子广播

充分因子广播利用梯度张量中的低秩结构，在计算图 [23]上进行数学等价的变换。充分因子是能够生成梯度张量的小型张量，通常通过外积生成。在 SFB 中，广播到算子副本的是充分因子，而非完整的梯度张量，因此有望减少通信时间。SFB 不会造成精度损失，也不会影响训练收敛。

Chilimbi 等人 [35]利用卷积神经网络最后几层中的低秩结构，主张仅将激活向量和误差梯度向量显式发送给参数服务器。Poseidon[36]通过广播充分因子，在工作节点之间同步梯度。这些方法仅支持矩阵乘法层，因此难以应用于包含新型算子的近期模型。这些方法不会自动启用，机器学习开发者需要选择应用这些优化的位置，以获得最佳效率。

## 3 动机与挑战

### 3.1 机遇

**部分数据并行**。现有框架要么在所有 GPU 上复制一个算子，要么在纯数据并行、模型并行或混合数据并行/模型并行中仅将其放置在一块 GPU 上 [8], [9], [10], [18], [19], [29]。在给定的 GPU 集群中，可以在一部分相邻 GPU 上复制某些操作，而不在其他 GPU 上复制，从而在 GPU 利用率和参数同步开销之间实现良好的权衡 [30]。

**拓扑感知自动设备放置**。在现有设计中，当设备拓扑发生变化时（例如，所使用的设备或设备之间的链路带宽发生变化），决策神经网络需要重新训练，因为它们未将设备拓扑作为其神经网络的输入，并且当设备拓扑变化时，其神经网络的结构也可能需要调整。例如，HeteroG [10] 中的图神经网络输出维度为 $M+4$，其中 $M$ 是设备数量；当设备数量发生变化时，其图神经网络需要采用新的输出维度重新训练。设计一个能够处理各种设备拓扑以及不同深度神经网络的策略制定框架，是一种更通用且更实用的解决方案。

**自动充分因子广播**。 随着 GPU 和 TPU 计算能力的提升，通信开销在深度神经网络训练中变得日益显著。在共享人工智能集群中使用分散于不同机器或机架上的零散资源进行训练时，最大限度地减少通信开销尤为重要。梯度压缩已被用于减小通信所需的张量大小。量化 [37] 和

稀疏化 [38]是最常用的梯度压缩方法 [16], , 但会造成精度损失。我们聚焦于 SFB，它能够在不影响训练收敛的情况下缩短通信时间。

### 3.2 挑战

**策略空间极其庞大**。假设我们要将由 $n$ 个算子组成的深度神经网络模型部署到一个包含 $m$ 个 GPU 的集群中。每个算子都可以在任意设备子集上进行副本部署，从而形成 $2^m - 1$ 种选择。此外，对于每个副本参数（假设共有 $n_p$ 个副本参数），我们需要在 AllReduce 和参数服务器架构之间选择一种方式来进行参数同步。总共需要探索 $(2^m - 1)^n + 2^{n_p}$ 种策略。

**基于强化学习的方法往往会出现过拟合**。强化学习（RL）已成功用于探索大型搜索空间 [31], [39]。然而，基于强化学习的方法往往会对训练数据过拟合，可能导致泛化性能较差 [28], [40], [41]。传统的正则化方法（例如数据增强）[40] 几乎无济于事，因为实际使用的模型和设备拓扑可能与训练时所用的截然不同。

**SFB 并非总是有益**。 SFB 的通信成本取决于副本数量和充分因子的大小（这与批大小密切相关）。对于每个梯度，都需要评估采用 SFB 还是梯度同步方法（AllReduce 或 PS）才能使通信量最小。

### 3.3 解决方案

为充分利用上述机会并应对这些挑战，我们提出了 TAG，这是一种自动深度神经网络部署框架，具有以下关键设计。

**基于运行时反馈的交互式策略优化**。GDP [9] 和 HeteroG [10] 等现有方案采用一次性生成策略的方式：其模型直接输出适用于整个图的策略。虽然结果令人印象深刻，但其决策过程难以解释和推理。相反，人类开发者通常会通过迭代和交互的方式优化分布式训练策略。开发者可能会先运行一个简单的策略，然后检查执行轨迹，找出该策略的瓶颈并据此改进。例如，如果某个策略导致一块 GPU 利用率低、空闲时间长，那么可以考虑将更多操作放到这块 GPU 上。[15] 的第 5 节给出了这种做法的具体示例。基于这一观察，我们在 TAG 中设计了一个自动化的交互式策略制定流程。它不再仅仅接收

将模型的原始特征作为输入并直接生成策略；TAG 会反复模拟策略的执行，收集相应的模拟执行轨迹，然后根据现有策略的运行时反馈尝试生成更好的策略。交互式策略探索过程还能帮助 TAG 为未见过的模型生成可行策略（例如，不会导致显存溢出的策略）。对于采用一次性生成策略的系统，如果生成的策略导致显存溢出，就需要用户介入调整参数并重试。通过交互式策略探索，遇到显存溢出时，TAG 会自动尝试采用更激进的模型并行来降低内存用量，直到找到可行的解决方案。

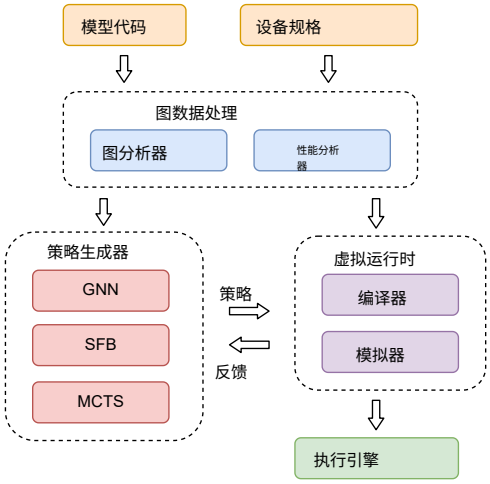



图1：TAG 的工作流程。

**结合学习与搜索**。基于学习的方法要求训练所用的数据集与实际使用场景遵循相同的分布，否则训练数据可能会过拟合 [40], [41]。然而，要收集能够代表“深度神经网络模型”分布的数据集非常困难。例如，PlaceTo [8]仅用 3 个模型进行训练，GDP [9] 基于 11 个模型进行学习，而 HeteroG [10] 则用 8 个模型进行训练。随着新的深度神经网络模型不断涌现，新模型可能与小型训练数据集中的模型截然不同。为缓解这一问题，我们认为，基于学习的方法应与一种能够稳健处理任意未见模型的搜索方法结合。在 TAG 中，我们将强化学习与蒙特卡洛树搜索结合起来，其中强化学习智能体使用图神经网络生成先验概率，以引导蒙特卡洛树搜索过程。

**计算图与设备拓扑的统一表示**。近期研究采用图神经网络，为不同的计算图生成分布式训练策略 [8], [9], [10]。图神经网络的参数维度不依赖于图中节点的数量，因此能够泛化到规模不同的图。然而，现有研究仅将计算图编码为图神经网络的输入，其设计在未经重新训练的情况下无法适应未见过的设备拓扑。为了使 TAG 能够泛化到不同的设备拓扑，我们使用包含计算节点和设备节点的异构图作为输入。这样便可以在统一的图中编码所有输入，并训练一个能够泛化到未见过的深度神经网络模型和设备拓扑的异构图神经网络。

**将 SFB 表述为优化问题**。SFB 处理参数的梯度，而这些梯度唯一的使用者是 ApplyGradient 算子。是否对某个梯度应用 SFB 是一个局部决策，与其他算子的策略无关。我们将每个梯度对应的 SFB 决策表述为一个混合整数优化问题，并可使用现成求解器高效求解。即使对于大型图，该优化问题的规模仍然很小，因为它只关注梯度周围的子图。

## 4 系统设计

TAG 的工作流程如图 1 所示。用户首先使用深度学习平台 API 构建单 GPU 深度神经网络计算图，例如使用 TensorFlow 提供的 API。该计算

图与设备规格一同输入 TAG，设备规格描述了分配用于训练该深度神经网络的可用 GPU。尽管本文重点关注分布式训练，TAG 同时支持训练和推理任务。对于训练任务，输入计算图应包含前向和反向操作（例如由 TensorFlow 的自动微分引擎生成）。对于推理任务，输入计算图仅包含前向传播中的操作。

TAG 首先从计算图和设备规格中提取必要特征。性能分析器会收集每种 GPU 上各个算子的计算时间，以及 GPU 之间的集合通信性能。利用这些数据，TAG 反复调用策略生成器和虚拟运行时，以寻找良好的部署策略。除了 Placeholder 等少数特殊算子外，TAG 不依赖任何领域知识，也不区分特定算子。因此，TAG 具有通用性，并且支持用户定义的算子。

策略生成器为给定的深度神经网络模型确定部署策略。在虚拟运行时模块中，编译器会相应地修改计算图，插入必要的辅助操作，以确保修改后的图与原始图等价。模拟器会模拟修改后计算图的执行，并估算训练时间，将其作为反馈提供给策略生成器，以生成新的、可能更优的策略。

### 4.1 图数据处理

#### *4.1.1* 图分析器

图分析器会构建原始计算图的内部表示，该表示独立于所使用的 API。

**简化图**。 删除不必要的节点，包括恒等算子、NoOp，以及未连接到主要优化算子（例如 GradientDescent、Adam）的悬空操作。这样可以在不改变语义的情况下缩小图的规模。

**标注张量拆分和拼接方法**。如果两个生产者-消费者算子分别在数量不同的设备上复制，则会添加拆分和拼接算子：复制张量会在发送给未复制的算子之前进行拼接；完整张量则可以拆分，并将不同部分发送给复制算子的不同副本。不过，只有部分操作可以在保持数学等价性的同时进行拆分和拼接。图分析器会为每个算子标注其可拆分性，供编译器后续使用。具体来说，图分析器会将每个算子归入以下类别之一。

- 可拆分并拼接。 此类别中的算子可以接受沿批次维度拆分的输入张量。输入拆分后，此类算子的输出也会被拆分，并可沿批次维度拼接以还原完整张量。此类算子包括逐元素算子（例如 AddN）、批处理 Conv2D 和 MaxPool2D 等。

- 可拆分并逐元素求和。此类别中的操作也可以接受拆分后的输入张量，但要还原完整的输出张量，需要对输出求和，而不是进行拼接。此类别通常包括生成梯度的操作，例如 Conv2DBackpropFilter。

- 其他。这些操作不接受拆分后的张量。如果生产者操作被复制，则在此类操作使用输入张量之前，必须先将其聚合，以还原完整张量。具体而言，TAG 将 ApplyGradient 算子归入此类别，以便在应用梯度到参数之前自动聚合梯度。

**对操作进行分组**。不同深度神经网络模型中的操作数量差异很大。例如，使用 TensorFlow Slim [42] 实现的 VGG 模型有 1169 个节点，而 BERT-Large 的典型实现有 26601 个节点。为了高效地为不同规模的模型生成策略，图分析器采用 METIS [43] 将一些紧密耦合的操作分组。我们使用 METIS 对计算图进行划分，将其分成不超过 60 个组，并尽可能减小割边上的张量大小，同时通过将平衡因子设为 2，使每个分区的总计算时间保持均衡。由于不同算子组的复制或放置策略可能不同，因此需要额外的通信来聚合和分发算子组边界上的张量。METIS 会尽量减小这些张量的大小，从而最大限度地降低通信开销。传递给策略生成器的图中，每个算子组都被视为一个节点。算子组数量越多，策略就越精细，但搜索时间也会延长。我们的实验发现，60 个组取得了良好的权衡。

#### *4.1.2* 性能分析器

为了测量每个算子的计算时间，性能分析器会在每种类型的 GPU 上运行单 GPU 模型训练。由于算子可能会被复制，因此我们需要了解不同批量大小下的计算时间。我们使用 60 以下的典型批量大小来分析算子执行时间。结果表明， [44] 当批量足够大时，计算时间几乎与批量大小呈线性关系。我们建立了一个线性模型，用于预测未进行性能分析的较大批量下的计算时间。对于即使在较小批量下也无法在单个 GPU 上容纳的大型模型，我们会手动对模型进行分区，并分别分析每个部分。

为了预测张量传输时间，性能分析器会测量 GRPC 传输（设备对之间）和 NCCL AllReduce 通信（不同设备组合之间）的性能。传输不同大小的随机 32 位浮点数数据（从 1KB 开始，每次翻倍，直到 1GB）。我们分别为 GRPC 传输和 AllReduce 通信构建分段线性回归模型。

### 4.2 策略生成器

策略生成器由三个组件构成。蒙特卡洛树搜索逐步为每个操作组生成放置和复制策略。在搜索过程中，利用异构图神经网络，根据融合计算图与设备拓扑图的图输入，为蒙特卡洛树搜索采样策略候选项提供先验概率。当蒙特卡洛树搜索生成复制参数的策略时，*SFB* 求解器 会判断是否可以应用 SFB 来减少通信。

策略生成器的输入包括计算图和设备图。前者包含 $N$ 个节点，每个节点代表一个操作组。后者包含 $M$ 个节点，每个节点表示一组同构 GPU，即一组 GPU

，这组 GPU 的类型相同，且任意两者之间的带宽相同。这通常对应一台配备多块同类型 GPU 的机器。

策略生成器会生成部署策略，其中包含算子放置以及每个复制算子的复制计划。放置 $P$ 是一个 $N \times M$ 二进制矩阵。$P_{i,\ j} = 1$ 如果第 i 个操作组被放置在第 j 个设备组中的一个或全部设备上（取决于复制计划）。复制计划 $O$ 是一个 $N \times 4$ 矩阵，其第 i 行对应于第 i 个操作组的 4 种复制选项的独热编码。$O_i$ 定义了当设备组包含多个设备时，如何将第 i 个操作组放置在相应的设备组上。TAG 中考虑的 4 种复制选项如下：

• **使用 `AllReduce` 进行复制**。 操作组会复制到设备组中的所有设备上，并沿批次维度均匀拆分输入张量。如果操作组产生梯度，则使用 AllReduce 操作进行同步。

• **使用参数服务器进行复制**。 除了使用参数服务器同步梯度之外，这与使用 AllReduce 进行复制类似。参数服务器以轮询方式从设备组中的 GPU 中选取。

• **复制**。 该操作组将被复制到设备组中的所有设备上。与复制不同，输入张量会广播到所有副本，因此所有设备上产生的梯度都相同，无需同步。

• **模型并行**。 操作组中的算子使用 METIS 划分为更小的组，并放置到设备组中的不同设备上。这有助于放置大型模型，否则这些模型会导致显存溢出（OOM）。

#### *4.2.1* 异构图神经网络

该图神经网络以异构图作为输入，其中包含两种类型的节点和三种类型的链路。第一种节点类型是计算节点，每个节点代表一组操作；另一种是设备节点，代表一个同构 GPU 组。三种链路类型包括：（i）连接两个计算节点的链路，对应计算图中的张量；（ii）连接设备节点的链路，代表网络链路或 PCI 交换机；（ iii ）连接计算节点和设备节点的链路，表示相应操作组在设备组中的具体放置。

图神经网络的输入特征包含四个部分：(1) 计算图和设备的原始特征，包括总计算时间（取不同设备上测得运行时间的平均值）、每个算子组的参数总大小、每个设备组中的 GPU 数量、每个 GPU 的内存容量、每个设备组中 GPU 之间的带宽、连接两个算子组的张量大小，以及每对设备组之间的组间带宽。(2) 策略编码，即所有算子组复制计划的独热编码，以及每条连接算子组与设备组的边所对应的二元边特征，用于指示前者是否部署在后者上。(3) 输入策略的运行时反馈，包括每个算子组执行的总工期、算子组执行结束到其输出张量开始传输之间的平均空闲时间、每个设备组的峰值内存使用量和空闲百分比，以及设备组之间每条链路的空闲百分比。(4) 搜索进度，即独热

表1：图神经网络输入特征。

| Type | 特征 |
|---|---|
| 操作节点 | 计算时间 |
| | 参数大小 |
| | 复制计划 |
| | 总工期 |
| | 输出传输前的空闲时间 |
| | 如果策略已确定 |
| | 如果接下来要生成策略 |
| 设备节点 | 组内 GPU 数量 |
| | 每个 GPU 的内存容量 |
| | 组内带宽 |
| | 峰值内存使用量 |
| | 空闲百分比 |
| 算子-算子边 | 张量大小 |
| 设备-设备 edge | 组间带宽 |
| | 空闲百分比 |
| 算子-设备边 | 放置 |

编码，用于指示哪些算子组的部署策略已确定，以及接下来将生成哪个算子组的策略。我们添加全连接层，将节点相关特征和边相关特征转换为固定长度 $f$ 的特征向量，再将其输入图神经网络，从而使嵌入长度在图神经网络卷积过程中保持不变。我们在表1中总结了输入特征。

我们采用一个 4 层图神经网络。每一层图神经网络都会通过 $h_u^{i+1} = \text{AGG}_{v \in \mathcal{N}(v)} \gamma_{etype} \cdot \sigma(W_{i,etype}(h_v^i \circ e_{uv}) + b_{i,etype})$ 转换上一层生成的嵌入。这里，$h_u^i$ 是节点 $u$ 在第 i 层的嵌入，$\mathcal{N}(v)$ 是输入异构图中节点 $v$ 的邻居集合，$e_{uv}$ 是 $u$ 与 $v$之间的边特征。$\circ$ 表示向量拼接。$W_{i、\ etype}$ 以及$b_{i、\ etype}$ 是第 i 层针对 $etype$ 的参数，$\sigma$ 是一个非线性函数。AGG 表示对邻居特征进行聚合。$h_u^0$ 初始化为节点 $u$的节点特征。$e$ 研究发现，深层图神经网络会遇到过平滑问题 [45], [46], ，也就是说，层数更深的图神经网络不一定表现更好。我们的经验也证实了这一现象，并发现 4 层在 TAG 中效果最佳。我们选择采用图注意力网络（GAT）中介绍的基于多头注意力的聚合方式 [47] ，因为它会为不同邻居分配不同权重；我们认为，某些邻居在确定操作组的最佳策略时更为重要，因此这种做法十分合适。$\gamma_{etype}$ 表示不同类型边的权重：对于连接相同类型节点的边，$\gamma_{etype}$ 设为 1；对于连接不同类型节点的边，则设为 0.1，从而平衡连接到一个节点的不同类型边。图 2 展示了图神经网络的结构。

图神经网络的输出包括算子组的嵌入 $E_{op}$ 和设备组的嵌入 $E_{dev}$。图神经网络还连接了一个精简的解码器，其中包含一个简单的全连接层，其输入是一个策略切片（$P_i$,$O_i$），其中包含第 i 个算子组的放置与复制计划，以及特征向量 $\sum_{j=1}^{M} E_{dev}[j] P_{i,\ j} \circ E_{op}[i] \circ O_i$ （包含图神经网络生成的嵌入）。该层会计算策略切片的得分。随后，一个 Softmax 算子会根据得分进一步生成第 i 个算子组所有可能策略切片的概率。这些概率用于引导蒙特卡洛树搜索的探索。

#### *4.2.2* 蒙特卡洛树搜索

MCTS 是一种最佳优先搜索算法，能够平衡利用与探索。在我们的搜索中，搜索树中的一个顶点

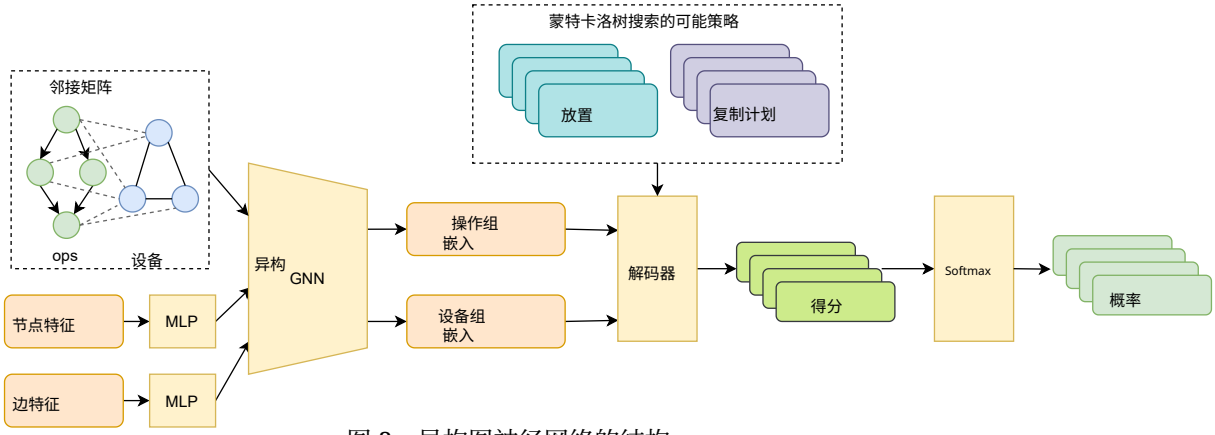



图 2：异构图神经网络的结构。

表示一个部分部署策略 $s$，一条边表示一个动作 $a$。部分策略包含尚未完成的放置与复制计划矩阵 $P$ 和 $O$，即已填入部分行的矩阵，这些行对应于某些算子组。$a$ 是要应用于下一个待处理算子组的部署策略。算子组按计算时间降序排列，因此计算开销最大的算子组将优先处理。每条边都会记录其访问次数 $N(s,a)$ 以及运行平均奖励 $Q(s,a)$。我们使用模拟器估算采用当前部署策略时深度神经网络图的执行时间，并计算其相对于基线策略（即采用基于 AllReduce 的参数同步的数据并行）的加速比，作为奖励。

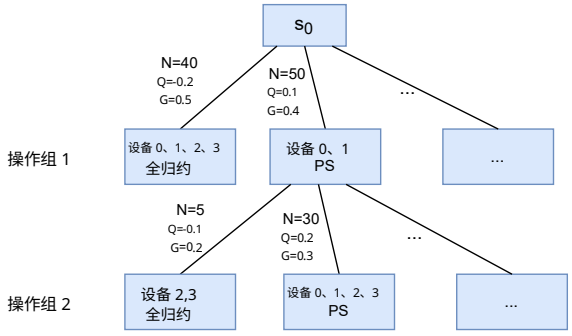



图3：蒙特卡洛树搜索的搜索树示例。

蒙特卡洛树搜索从一个空策略 $s_0$ 开始，并通过反复执行以下步骤逐步构建搜索树：

- 选择：从搜索树的根节点开始，持续选择子顶点进行遍历，直至到达叶节点（尚未扩展的顶点，或所有算子组均已获得完整部署策略的顶点）。在每个顶点 $s$，选择 PUCT 分数最高的边 $(s,a)$ [22]，作为下一步遍历的边，定义如下：

$$U(s,a) = Q(s,a) + cG(s,a)\frac{\sqrt{\sum_{a'\in\phi} N(s,a')}}{1+N(s,a)}$$

其中，$c$ 是一个系数，$G(s,a)$ 是图神经网络生成的先验概率，$\phi$ 表示 $s$ 的所有子顶点中的动作。PUCT 分数会优先选择最有希望进行探索的策略。

- 扩展与评估：到达叶节点时，我们使用模拟器评估该顶点的奖励 $r$，即其策略相对于基线（数据并行）所实现的训练加速比。[2] 如果导致显存溢出错误，则将奖励设为 $-1$。随后，我们将子树扩展一个层级，枚举下一个算子组的可能策略，并从图神经网络中获取这些策略的先验概率 $G(s,a)$。

- 反向传播：在叶节点获得奖励 $r$ 后，沿着从根节点到叶节点的路径更新运行平均奖励 $Q(s,a)$ 和访问次数 $N(s,a)$，例如，$Q(s,a) := Q(s,a) + \frac{1}{N(s,a)} \cdot r$。

图 3 展示了蒙特卡洛树搜索的一个示例搜索树。根节点是空策略 $s_0$。每一层都包含一个操作组的放置与复制计划。在每次迭代中，根据选择策略从根节点遍历搜索树，直到叶节点。然后扩展一层并评估奖励，以更新路径上的 $Q$ 和 $N$。

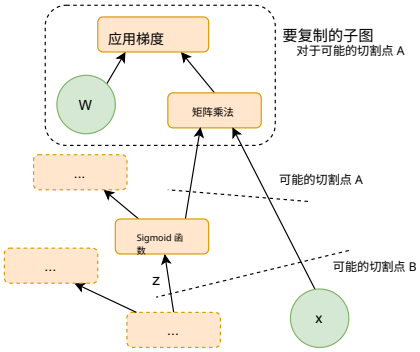
我们选择蒙特卡洛树搜索而非其他搜索方法，原因如下：首先，它可以自然地与一种基于强化学习的方法（用于更新图神经网络模型参数）集成，这对于泛化能力十分重要。在每一步图神经网络训练中，我们随机选择一个深度神经网络图和一个设备拓扑。我们运行蒙特卡洛树搜索，并收集访问次数至少为800的顶点处的选择概率 $\pi(s)$ =Softmaxln $N(s)$，其中$N(s)$ 是一个向量，包含 $s$所有子顶点的访问次数。随后更新参数 $\theta$，以最小化图神经网络生成的先验概率 $G_\theta(s,\ a)$ 与蒙特卡洛树搜索的选择概率 $\pi(s,\ a)$ 之间的交叉熵。其次，蒙特卡洛树搜索会逐步构建策略。在每个顶点处，我们评估部分策略，并利用运行时反馈帮助图神经网络更好地预测后续策略。

### *4.2.3* 充分因子广播

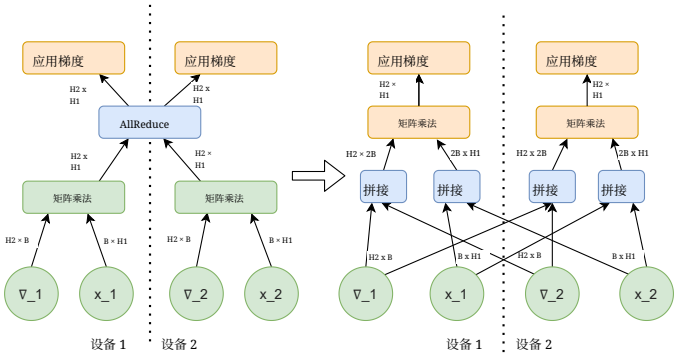
当一个参数在多个设备上复制（并配有相应算子）时，需要在这些设备之间同步其梯度。通常通过插入 AllReduce 或 PS算子来实现。另一种方法是，有些梯度可以根据较小尺寸的张量计算得出。例如，一个非满秩梯度矩阵可以表示为两个较小矩阵的乘积（这两个小矩阵就是充分因子）。我们的 SFB 求解器旨在自动查找图中数学等价的替代方案，从而有望减少通信。

图 4(b) 展示了将 SFB 应用于矩阵乘法算子的示例，该算子可见于许多全连接层中的

2.对于尚未确定部署策略的算子组，我们采用其中计算开销最大的算子组的策略。




(a) SFB 的示例割。




(b) (a) 中割 A 的 SFB 实现。

图 4：充分因子广播示例。

深度神经网络中。当矩阵乘法被复制到多个设备上时，我们不使用 AllReduce 或参数服务器来同步生成的梯度，而是将充分因子 $\nabla$ 和 $x$广播到所有设备，使每台设备上的矩阵乘法算子都能重建相同的梯度。在 TAG 中，这相当于将相应算子的部署策略设为“复制”。这会将通信数据总量从梯度大小 $H_2 \times H_1$变为充分因子的大小$2(H_2 \times B + B \times H_1)$，其中 $H_1$ 和 $H_2$ 分别是全连接层的输入和输出特征长度， $B$ 是批大小。我们的 SFB优化会识别出采用这种变更能够带来收益的梯度。

DNN 计算图包含用于更新模型中参数的优化器操作。考虑这样一个操作 $l$（例如图 4 中的应用梯度）。它的一个输入操作 $g$ （例如图 4 中的矩阵乘法）会生成该操作 $l$ 更新的参数梯度。如果包含 $g$ 的算子组由蒙特卡洛树搜索进行复制，并且张量 $(g,\ l)$ 需要在所有副本设备之间同步，我们会检查是否可以应用 SFB 来减少通信。请注意，蒙特卡洛树搜索也可以直接生成“复制”选项，以便在某些算子组上启用 SFB。这里，我们会再次检查蒙特卡洛树搜索已为其生成复制选项的算子组是否有应用 SFB 的机会。其理由在于，蒙特卡洛树搜索生成的是算子组级别的策略，而 METIS 确定的组边界很少是 SFB 的最佳割。

对于复制操作组中的每个梯度张量，我们求解以下优化问题，以确定可复制的子图（例如图 4(a) 中的虚线框）。由于充分因子包含用于计算相应梯度的所有输入，因此它们构成一个割，将子图与计算图的其余部分分隔开来。例如，在图 4(a) 中， $\nabla$ 和 $x$ 是一组充分因子

表2：SFB 优化中的符号。

| | |
|---|---|
| $E$ | 操作组内张量的集合 |
| $V$ | 操作组中的操作集合 |
| $l$ | 一个优化器算子 |
| $g$ | 一个为 $l$生成梯度的算子 |
| $D$ | 拥有 $g$副本的设备数量 |
| $T_i$ | 算子 $i$的计算时间 |
| $L_{ji}$ | 张量的大小 $(j,\ i)$ |
| $\tau$ | 设备 $D$ 之间的瓶颈带宽 |
| $\alpha_i$ | 是否复制算子 $i$ |
| $b_{ji}$ | 张量 $(j,\ i)$ 是否位于割中 |

，用于计算 MatMul 产生的梯度，并且它们构成一个割；只需基于 $\nabla$ 和 $x$运行子图，就可以计算 $W$ 的梯度。现有研究 [23], [36]仅考虑对两个向量执行矩阵乘法的操作的 SFB，即由两个向量的外积构成的梯度；只要能够找到这样的割，我们的方法就能识别其他情况。

$$\min\ (D-1)\sum_{i\in V}\alpha_i T_i + D(D-1)\sum_{(j,i)\in E} b_{ji}\frac{L_{ji}}{\tau} - 2\alpha_g\frac{D-1}{D}\frac{L_{gl}}{\tau}$$

$$\begin{aligned}\text{subject to:}\quad & \alpha_k \le \sum_{(k,i)\in E}\alpha_i, \quad \forall k \in V\setminus\{l\}\\ & b_{ji} \ge \alpha_i - \alpha_j, \quad \forall (j,i)\in E\\ & \alpha_i \in \{0,1\}, \quad \forall i \in V\\ & b_{ji} \in \{0,1\}, \quad \forall (j,i)\in E\end{aligned}$$

符号说明见表2。 $\alpha_i$是主要决策：$\alpha_i = 1$表示将算子 $i$ 的复制选项从“使用 AllReduce 复制”或“使用 PS 复制”更改为“复制”； $\alpha_i = 0$则表示其他情况。 $b_{ji}$ 指定张量 (j,i) 是否位于用于划分包含算子 $i$ 的子图的割中。

目标函数的第一项是因复制而增加的计算时间。由于我们会在放置了算子副本的所有设备上复制这些算子，因此与不进行复制相比，每台设备都需要多处理 $D-1$ 次这些算子。第二项减去第三项表示由此产生的额外通信时间。对于每个由复制算子生成、并被重复算子使用的张量，都需要进行 $D(D-1)$ 次张量传输，才能将其广播到参与复制的所有设备。如果不进行复制，则可以使用 AllReduce 或 PS 方法同步梯度；第三项则描述了使用环形AllReduce 时的通信时间（此处以此为例，其他 AllReduce 算法和 PS 的情况也可相应地进行描述）。第一个约束规定，只有当某个算子的一个使用者算子位于重复子图中时，该算子才会被纳入其中，否则它与梯度无关。第二个约束确保当 $i$ 被复制而 $j$ 未被复制时，张量 $(j,i)$ 位于割中。这是一个与最小割问题类似的整数线性规划，但在割的一侧额外加入了节点权重。我们使用 Cbc [48]求解器来求解该问题。

### 4.3 虚拟运行时

虚拟运行时无需在物理集群上实际运行策略，即可对其进行评估。它还会根据策略生成分布式训练图，该图随后可以

由执行引擎（例如 TensorFlow）加载并执行。

#### *4.3.1* 编译器

编译器以 DNN 计算图和策略生成器找到的部署策略 $(P,\ O)$ 为输入。它应用该策略，并生成修改后的计算图，供模拟器或实际执行引擎（例如 TensorFlow）使用。无论采用何种部署策略，编译器都会自动插入必要的辅助操作，以确保修改后的图与原始图等效。它首先根据放置 $P$ 将操作映射到设备，然后在以下情况下插入辅助操作：

- 当某个操作被复制，而其输入张量未被复制时，会插入Split操作来拆分输入张量，再将其馈送给该操作的各个副本。
- 当某个操作未被复制，而其输入张量被复制时，会添加拼接或 AddN 操作来聚合输入张量；对于标记为“可通过拼接拆分”和“可通过逐元素求和拆分”的父操作，分别采用拼接和 AddN 操作。
- 当操作和输入张量都被复制，但复制到的设备数量不同时，会同时插入拼接和 Split操作，以调整副本数量。
- 当参数被复制时，会根据复制选项插入 AllReduce 或 AddN 算子。

#### *4.3.2* 模拟器

该模拟器实现了一种与 TensorFlow 默认调度器类似的操作调度算法。它为每个设备设置一个 FIFO队列。当某个算子的所有输入张量都准备就绪时，该算子便会加入队列。每个设备都会独立地使用经过性能分析的数据，模拟执行其任务队列中的操作，并报告每个算子的完成时间。

该模拟器使用引用计数来跟踪张量的生命周期，并估算每个设备上的峰值内存使用量。算子执行完成后，其输出张量会计入相应设备的内存使用量。当所有使用某个张量的算子都执行完毕后，该张量便被视为已释放，并从设备内存使用量中移除。

## 5 实现与评估

### 5.1 实现

我们在 TensorFlow 1.14 上将 TAG 实现为一个 Python模块。对于操作分组，我们使用张量大小作为边权重，并在运行 METIS 时使用计算时间作为节点平衡约束。我们在实验中将默认分区数设为 60。性能分析器使用跟踪选项运行 TensorFlow，以收集不同批量大小下每个操作的计算时间。对于每个批量大小，我们对每个模型进行 5 次性能分析，以获得平均时间。

我们的异构图神经网络基于 DGL [49]实现。我们扩展了 DGL 中的 GAT [47] 实现，使其支持异构图和边特征。该图神经网络包含 55MB 的参数。我们使用 Cbc [48] 求解 SFB 优化问题。在实验中，我们发现它能够在数百毫秒内可靠地求解整数优化问题。

表3：基准深度神经网络模型。

| 模型 | 批大小 | 操作数 | 参数大小 |
|---|---|---|---|
| InceptionV3 [2] | 96 | 5312 | 90M |
| ResNet101 [1] | 96 | 7951 | 169M |
| VGG19 [51] | 96 | 1169 | 548M |
| Transformer [52] | 480 | 16859 | 407M |
| BERT-Small [6] | 96 | 5061 | 98M |
| BERT-Large [6] | 16 | 26601 | 2313M |

### 5.2 实验设置

**硬件**。 我们在两个集群上进行实验。第一个集群是一个由 7 台物理机器组成的本地集群（测试平台）：其中一台配备了 4 个 NVIDIA Tesla V100 32GB GPU；四台各配备 2 个 NVIDIA GTX 1080Ti GPU；另外两台各配备 2 个 NVIDIA Tesla P100 GPU。第一台机器配备 NVLink，其他机器则使用 PCIe。这些机器连接到一个 100Gbps 交换机。第二个集群位于公有云（云）上，由 6 台机器和 32 个 GPU 组成。其中两台机器各配备 8 个 NVIDIA Tesla V100 16GB GPU，另外 4 台各配备 4 个 NVIDIA Tesla T4 GPU。这些机器通过 10Gbps 带宽互连。

**基准测试**。 我们使用 6 种具有代表性的深度神经网络模型进行实验，涵盖图像分类和神经语言处理任务，如表3所示。实验中使用 Adam [50] 优化器。

**图神经网络训练**。 我们在策略生成器中使用 6 种深度神经网络模型、测试平台的设备拓扑以及随机生成的 100 种设备拓扑作为输入来训练图神经网络。随机设备拓扑的生成方式为：机器数量随机，每台机器配备 $[1, 6]$, $[1, 8]$ 个 GPU；GPU 类型从 3 种类型中随机选择；机器内带宽介于 $[64, 160]$ Gbps 之间（用于模拟是否存在 NVLink），机器间带宽则在 $[20, 50]$ Gbps 范围内（用于体现不同的机器位置）。图神经网络训练约需 2 天才能收敛。

**基线方法**。 我们将 TAG 与以下基线方法进行比较。**(1) DP-NCCL**：使用 NCCL [13]AllReduce 进行参数同步的数据并行。该方法广泛用于分布式训练，并内置于大多数深度学习框架中 [24],[25], [26]。我们在 TensorFlow 上使用标准的图内复制实现 DP-NCCL。**(2) DP-NCCL-P**：一种数据并行方法，根据设备计算能力的倒数为设备分配批大小。**(3) Horovod** [12]：一种数据并行训练框架（在我们的实验中与 TensorFlow 一起使用），包含重叠 AllReduce 与反向计算等优化。**(4)FlexFlow** [30]：一种支持 SOAP 维度并行化的分布式深度学习框架。FlexFlow 假设集群是同构的。我们使用其 Keras API 在该框架上实现了 VGG19、ResNet101 和 InceptionV3。由于缺少注意力层实现，我们无法在 FlexFlow 上实现所有基准模型。我们将其马尔可夫链蒙特卡洛搜索迭代次数设为 100000。**(5)HDP** [18]：一种异构性感知分层设备放置系统，使用强化学习联合学习分组和设备分配。**(6) Post** [53]：一种采用交叉熵最小化和近端策略优化的设备放置系统。**(7)PlaceTo** [8]：一种使用图神经网络和强化学习的设备放置系统。

表4：TAG 生成的部署策略。

| 模型 | 复制 | | | 通信 | |
|---|---|---|---|---|---|
| | *V100* | *1080Ti* | *P100* | *PS* | *AR* |
| InceptionV3 | 4.0 | 8.0 | 0.0 | 100% | 0% |
| VGG19 | 4.0 | 2.1 | 0.0 | 100% | 0% |
| ResNet101 | 4.0 | 8.0 | 4.0 | 67% | 23% |
| Transformer | 4.0 | 8.0 | 0.0 | 1.7% | 98.3% |
| Bert-Small | 3.9 | 0.1 | 0.1 | 15.0% | 85.0% |
| Bert-Large | 0.6 | 0.5 | 0.1 | 3.5% | 0% |

**(8) GDP** [9]：一种使用图神经网络和 Transformer 模型的设备放置系统。**(9) Baechi** [54]：一种算法式设备放置系统。我们仅将其与 *mSCT* 算法进行比较，因为据报道该算法优于论文中提出的另外两种算法。**(10) HeteroG** [10]：一种支持异构集群的先进系统，使用图神经网络进行算子部署决策。它支持与 TAG 类似的决策空间，但仅假设一种给定的设备拓扑，并将一个算子复制到所有设备上，或将其放置在单个设备上。我们使用测试平台的设备拓扑，在所有基准模型上训练 HeteroG。

部分基线系统并未开源，且重新实现并非易事。我们采用相关工作中使用的评估方法 [10], [54] ，并比较这些基线方法相较于专家策略所报告的改进效果。

### 5.3 异构集群上的训练加速

我们首先比较 TAG 与开源基线方法在我们的测试平台上产生的平均每次迭代训练时间（第5.2节）。图5显示，TAG 在所有模型上的表现都优于基线方法：与 DP-NCCL 相比，加速比为 8%–456%；与 DP-NCCL-P 相比，为 1%–391%；与 Horovod 相比，为 11%–381%；与 HeteroG 相比，为 4%–186%。请注意，HeteroG 是针对这一设备拓扑从头开始训练的。训练 VGG19 时，相较于 DP-NCCL 获得的加速效果最佳：VGG19 的参数数量相对较多，通信往往成为瓶颈；使用 DP-NCCL 时，必须等到最慢的设备就绪后才会进行参数同步，这会大幅拖慢训练。另一方面，ResNet101 等模型的参数较少，但计算密集度更高；DP-NCCL 能充分利用所有 GPU，因此 TAG 和 HeteroG 几乎找不到更好的策略。由于通信经过高度优化，Horovod 在大多数情况下优于 DP-NCCL，但在异构集群中的加速比非常有限。DP-NCCL-P 能平衡不同显卡的计算工作负载，在异构集群中的表现优于 DP-NCCL。然而，由于两个原因，整体提升非常有限。首先，其通信时间与 DP-NCCL 相同：两种方法都通过 AllReduce 在所有设备间同步所有参数。其次，不同算子的计算特性各不相同。不同算子对应的设备间最优批次大小分布也不同。FlexFlow 会找到混合策略，为不同层选择不同的副本数量和放置方案。然而，由于 FlexFlow 未考虑计算设备之间的异构性，它会给速度较慢的显卡分配过多工作负载，导致性能不理想。

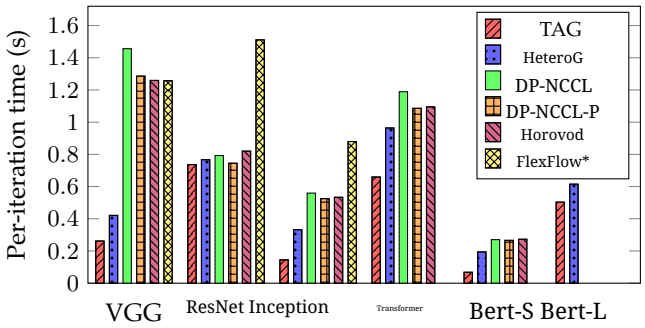



图5：每次迭代训练时间。DP-NCCL、DP-NCCL-P 和 Horovod 在使用 BERT-Large 时会导致显存溢出。*FlexFlow 基于 Legion 实现 [55]，而其他系统基于 TensorFlow 实现 [24]。

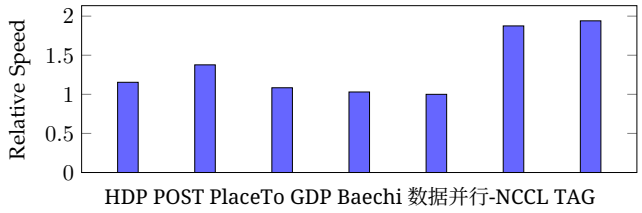



图6：同构集群上 InceptionV3 的训练速度。速度相对于人工专家策略计算。

表4详细列出了 TAG 生成的策略，包括各类算子平均复制到多少个 GPU 上，以及使用参数服务器或 AllReduce 进行梯度同步的比例。对于 ResNet101，TAG 会将所有算子复制到所有设备上；对于其他模型，4 张 P100 GPU 很少被利用，因为使用这些设备带来的收益不足以抵消额外的通信开销。我们观察到，训练大多数模型时，参数同步采用参数服务器与 AllReduce 混合的方式；没有选择“复制”选项，因为它主要适用于较小的批量大小（如第5.6节将介绍的情况），而本实验使用的批量大小相对较大。

### 5.4 同构集群上的训练加速比

我们将 TAG 与更多基线方法进行比较，其中包括未提供开源实现的方法；具体做法是遵循相关工作 [10], [54]中的评估方法，比较各方法相对于人工专家策略的已报告加速比。为确保公平比较，我们在与这些工作相似的硬件配置下开展实验，即在同一台机器上使用两块 V100 GPU 的同构集群。我们使用 InceptionV3 作为基准模型，因为它是所有这些工作评估中唯一共同采用的模型。对于该基准模型，我们采用现有研究 [18], [53] 中相同的专家策略。如图6所示，TAG 的表现优于所有基线方法，提升幅度为 3%–94%。

### 5.5 运行时反馈特征的有效性

输入到我们的图神经网络中的四部分特征（第 4.2.1 节）中，第三部分包含若干由模拟器提供的特征。其他利用模拟器驱动图神经网络学习的研究（例如 FlexFlow [30]和 HeteroG [10]）主要只利用模拟器生成的执行时间，而我们则使用

表5：应用和未应用充分因子广播时的每次迭代训练时间（秒）。

| 模型 | DP-NCCL | | | TAG | | | FlexFlow |
|---|---|---|---|---|---|---|---|
| | 未使用 ***SFB*** | 使用 ***SFB*** | 加速比 | 不使用 ***SFB*** | 使用 ***SFB*** | 加速比 | |
| InceptionV3 | 0.0898 | 0.0452 | 98.7% | 0.0775 | 0.0420 | 84.5% | 0.0631 |
| ResNet101 | 0.1122 | 0.0854 | 31.4% | 0.0508 | 0.0490 | 3.8% | 0.0552 |
| VGG19 | 0.1195 | 0.1192 | 0.3% | 0.1167 | 0.1169 | -0.2% | 0.1083 |
| Transformer | 0.1199 | 0.0455 | 163.5% | 0.0521 | 0.0451 | 15.5% | N/A |
| BERT-Small | 0.0618 | 0.0546 | 13.2% | 0.0597 | 0.0535 | 11.6% | N/A |
| BERT-Large | 0.4576 | 0.4317 | 6.0% | 0.4331 | 0.4266 | 1.5% | N/A |

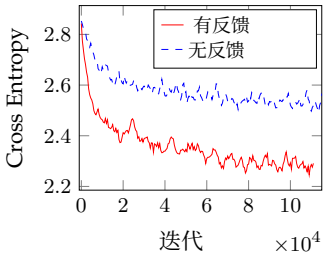



图8：图神经网络的损失曲线。

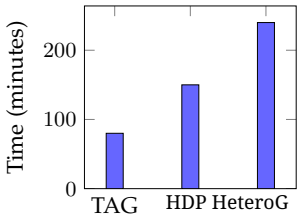



图8：在未见过的设备拓扑上生成策略的开销。

表6：TAG 选择复制的前 5 个操作。

| 操作 | 数量 |
|---|---|
| 重塑 | 341 |
| 矩阵乘法 | 336 |
| 转置 | 89 |
| Conv2DBackpropFilter | 66 |
| Add | 26 |

由模拟器估计的多维信息作为图神经网络输入。为了评估此类特征的效果，我们在有无这些特征的情况下训练图神经网络。图7表明，运行时反馈特征显著提升了图神经网络的学习效果。

### 5.6 充分因子广播的效果

为揭示SFB的潜在优势，我们在两台机器上进行了该实验，每台机器均配备一块1080Ti GPU。所有模型均使用批大小为4。表5比较了启用SFB前后，采用DP-NCCL和TAG所实现的每次迭代训练时间。我们还将FlexFlow纳入作为参考。当将SFB应用于DP-NCCL时，我们求解SFB优化问题，以找到每个梯度周围的一个子图，并用“复制”选项替代该梯度的AllReduce同步。SFB在训练InceptionV3和Transformer时带来了显著加速比，这表明这些模型中存在更多低秩结构。由于TAG还采用了其他策略来缓解通信开销，例如混合使用PS和AllReduce，因此TAG的总通信时间更短，与DP情形相比，在TAG中应用SFB所获得的加速比也更小。

我们在表6中汇总了6个深度神经网络模型中通过SFB优化复制次数最多的前5个操作。计数表示相应操作通过SFB优化被复制的总次数。TAG能够识别矩阵乘法之外的SFB机会。

### 5.7 对未见过的设备拓扑的泛化能力

为了评估我们的图神经网络模型的泛化能力，我们随机生成了100 种未见过的设备拓扑（生成方式与设备拓扑的生成方式（如第5.2节所述，这些拓扑用于图神经网络训练），并使用模拟器评估 TAG 为在每种未见过的拓扑上训练一个深度神经网络（从6个模型中随机选择）而生成的策略。

表7：获得优于DP-NCCL的策略所需的蒙特卡洛树搜索平均迭代次数。

| 模型 | 纯蒙特卡洛树搜索 | TAG |
|---|---|---|
| InceptionV3 | 66.0 | 9.5 |
| ResNet101 | 73.4 | 4.6 |
| VGG19 | 56.6 | 17.7 |
| Transformer | 145.0 | 121.8 |
| Bert-Small | 97.8 | 7.9 |

表8：相较于 DP-NCCL 的平均加速比。TAG：使用所有模型训练。TAG–：使用其他深度神经网络训练，并为一个留出模型生成策略。

| 模型 | 测试平台 | | 云 | |
|---|---|---|---|---|
| | ***TAG*** | ***TAG–*** | ***TAG*** | ***TAG–*** |
| InceptionV3 | 456.1% | 456.1% | 117.7% | 117.7% |
| ResNet101 | 7.7% | 7.1% | 12.2% | 10.4% |
| VGG19 | 286.2% | 213.6% | 43.6% | 43.6% |
| Transformer | 80.3% | 80.3% | 15.0% | 13.4% |
| Bert-Small | 298.4% | 279.6% | 84.5% | 84.5% |

我们统计 TAG 找到优于 DP-NCCL 的部署策略所需的蒙特卡洛树搜索迭代次数。我们还将该次数与不使用图神经网络提供的先验概率、而是使用均匀分布概率的纯蒙特卡洛树搜索所需的迭代次数进行比较。从表7可以看出，借助图神经网络提供的先验概率，TAG 能够迅速找到优于 DP-NCCL 的策略，而纯蒙特卡洛树搜索则需要更多迭代。

### 5.8 对未见过的计算图的泛化能力

我们使用表3中的5个模型训练 TAG，然后为留出模型生成策略（在6个深度神经网络中轮换留出模型）。我们在测试平台和云上都进行了这项实验。如表8所示，为未见过的模型生成的策略仅比为训练集中的模型生成的策略略差。

### 5.9 TAG 的开销

我们将 TAG 的开销与两种基于学习的方法 HDP [18] 和 HeteroG [10]进行了比较。如图8所示，TAG 比 HDP 快 87.5%，比 HeteroG 快 2 倍。得益于对未见设备拓扑的泛化能力，TAG 只需运行蒙特卡洛树搜索和图神经网络推理即可生成策略，而 HeteroG 则需要从头开始训练。HDP 在搜索过程中持续在真实集群上评估策略，因此会产生较大的开销。

## 6 结论

本文提出了 TAG，这是一种自动化 DNN 部署系统，可加速分散资源上的分布式训练。TAG 将 DNN 计算图和设备拓扑图同时作为图神经网络的输入，并将图神经网络与蒙特卡洛树搜索相结合，以确定优化后的部署策略。通过自动进行部分复制和充分因子广播，TAG 能够更好地利用异构资源，并降低深度神经网络训练的通信开销。在我们的实验中，与具有代表性的现有方案相比，TAG 的加速比最高可达 4.56 倍。TAG 通用且高效，无需重新训练即可为未见设备拓扑和深度神经网络模型生成优质策略。

未来的一个方向是扩展 TAG 以支持流水线并行，即扩展复制计划，加入“流水线”选项。图编译器需要在分布式图中添加适当的控制依赖，使算子能够在同一流水线阶段处理不同的微批次，从而实现高效流水线处理。


## 致谢

本研究部分由阿里巴巴集团通过阿里巴巴创新研究（AIR）计划以及香港研究资助局资助，资助合同编号为 HKU 17204619 和 17208920。


## 参考文献


[1] K. He、X. Zhang、S. Ren 和 J. Sun，“用于图像识别的深度残差学习”，载于电气电子工程师学会计算机视觉与模式识别会议论文集，2016年，第 770–778 页。 [2] C. Szegedy、V. Vanhoucke、S. Ioffe、J. Shlens 和 Z. Wojna，“重新思考用于计算机视觉的 Inception 架构”，载于电气电子工程师学会计算机视觉与模式识别会议论文集，2016年，第 2818–2826 页。 [3] T. B. Brown、B. Mann、N. Ryder、M. Subbiah、J. Kaplan、P. Dhariwal、A. Neelakantan、P. Shyam、G. Sastry、A. Askell 等，“语言模型是少样本学习者”，*arXiv* 预印本 *arXiv:2005.14165*，2020年。 [4] A. Chowdhery、S. Narang、J. Devlin、M. Bosma、G. Mishra、A. Roberts、P. Barham、H. W. Chung、C. Sutton、S. Gehrmann 等，“PaLM：借助 Pathways 扩展语言建模”，*arXiv* 预印本 *arXiv:2204.02311*，2022年。 [5] J. Wang、P. Huang、H. Zhao、Z. Zhang、B. Zhao 和 D. L. Lee，“用于阿里巴巴电子商务推荐的十亿规模商品嵌入”，载于第 *24* 届 *ACM SIGKDD* 国际知识发现与数据挖掘会议论文集，2018年，第 839–848 页。 [6] J. Devlin、M.-W. Chang、K. Lee 和 K. Toutanova，“BERT：用于语言理解的深度双向变换器预训练”，*arXiv* 预印本 *arXiv:1810.04805*，2018年。 [7] J. Lee、W. Yoon、S. Kim、D. Kim、S. Kim、C. H. So 和 J. Kang，“BioBERT：用于生物医学文本挖掘的预训练生物医学语言表示模型”，《生物信息学》，第 36 卷，第 4 期，第 1234–1240 页，2020年。 [8] R. Addanki、S. B. Venkatakrishnan、S. Gupta、H. Mao 和 M. Alizadeh，“PlaceTo：学习可泛化的分布式机器学习设备放置算法”，*arXiv* 预印本 *arXiv:1906.08879*，2019年。 [9] Y. Zhou、S. Roy、A. Abdolrashidi、D. Wong、P. C. Ma、Q. Xu、M. Zhong、H. Liu、A. Goldie、A. Mirhoseini 等，“GDP：面向数据流图的通用设备放置”，*arXiv* 预印本 *arXiv:1910.01578*，2019年。 [10] X. Yi、S. Zhang、Z. Luo、G. Long、L. Diao、C. Wu、Z. Zheng、J. Yang 和 W. Lin，“异构 GPU 集群中的分布式训练部署优化 ”，载于第 *16* 届新兴网络实验与技术国际会议论文集，2020年，第 93–107 页。 [11] H. Zhang、Y. Li、Z. Deng、X. Liang、L. Carin 和 E. Xing，“Autosync：学习如何同步数据并行分布式深度学习”，神经信息处理系统进展，第 33 卷，2020年。

[12] A. Sergeev 和 M. Del Balso，《Horovod：在 TensorFlow 中实现快速且简便的分布式深度学习》，arXiv 预印本 arXiv:1802.05799，2018年。 [13] S. Jeaugey，《NCCL 2.0》，载于 GPU技术大会（GTC），2017年。 [14] M. Li、D. G. Andersen、J. W. Park、A. J. Smola、A. Ahmed、V. Josifovski、J. Long、E. J. Shekita 和 B.-Y. Su，《使用参数服务器扩展分布式机器学习》，载于第11届 {USENIX}操作系统设计与实现研讨会（{ } OSDI），*14* 2014， 583 598 第 – 页。 [15] Y. Jiang、Y. Zhu、C. Lan、B. Yi、Y. Cui 和 C. Guo，《用于加速异构 GPU/CPU 集群中分布式 {深度神经网络} 训练的统一架构》，载于第14届 {USENIX} 操作系统设计与实现研讨会（{OSDI} *20*），2020年，第463–479页。 [16] Y. Lin、S. Han、H. Mao、Y. Wang 和 W. J. Dally，《深度梯度压缩：降低分布式训练的通信带宽》，arXiv 预印本 arXiv:1712.01887，2017年。 [17] S. Kim、G.-I. Yu、H. Park、S. Cho、E. Jeong、H. Ha、S. Lee、J. S. Jeong 和 B.-G. Chun，《Parallax：深度神经网络的稀疏性感知数据并行训练》，arXiv 预印本 arXiv:1808.02621，2018年。 [18] A. Mirhoseini、A. Goldie、H. Pham、B. Steiner、Q. V. Le 和 J. Dean，《用于设备放置的分层模型》，载于国际学习表征会议，2018年。 [19] A. Mirhoseini、H. Pham、Q. V. Le、B. Steiner、R. Larsen、Y. Zhou、N. Kumar、M. Norouzi、S. Bengio 和 J. Dean，《使用强化学习进行设备放置优化》，载于国际机器学习会议。PMLR，2017年，第2430–2439页。 [20] C. Zhang、D. Song、C. Huang、A. Swami 和 N. V. Chawla，《异构图神经网络》，载于第25届 ACM SIGKDD 国际知识发现与数据挖掘会议，2019年，第793–803页。 [21] L. Kocsis 和 C. Szepesv´ari，《基于赌博机的蒙特卡洛规划》，载于欧洲机器学习会议。施普林格，2006年，第282–293页。 [22] D. Auger、A. Couetoux 和 O. Teytaud，《具有多项式探索—一致性的连续上置信树》，载于欧洲机器学习与数据库知识发现联合会议。施普林格，2013年，第194–209页。 [23] P. Xie、J. K. Kim、Y. Zhou、Q. Ho、A. Kumar、Y. Yu 和 E. Xing，《通过充分因子广播实现分布式机器学习》，arXiv 预印本 arXiv:1511.08486，2015年。 [24] M. Abadi、P. Barham、J. Chen、Z. Chen、A. Davis、J. Dean、M. Devin、S. Ghemawat、G. Irving、M. Isard 等，《TensorFlow：用于大规模机器学习的系统》，载于第12届 {USENIX} 操作系统设计与实现研讨会（{OSDI} *16*），2016年。 [25] A. Paszke、S. Gross、S. Chintala、G. Chanan、E. Yang、Z. DeVito、Z. Lin、A. Desmaison、L. Antiga 和 A. Lerer，《PyTorch 中的自动微分》，2017年。 [26] T. Chen、M. Li、Y. Li、M. Lin、N. Wang、M. Wang、T. Xiao、B. Xu、C. Zhang 和 Z. Zhang，《MXNet：面向异构分布式系统的灵活高效机器学习库》，arXiv 预印本 arXiv:1512.01274，2015年。 [27] S. Pal、E. Ebrahimi、A. Zulfiqar、Y. Fu、V. Zhang、S. Migacz、D. Nellans 和 P. Gupta，《优化深度学习训练的多 GPU 并行化策略》，IEEE Micro，第39卷，第5期，第91–101页，2019年。 [28] Y. Zhou、S. Roy、A. Abdolrashidi、D. Wong、P. Ma、Q. Xu、H. Liu、M. P. Phothilimtha、S. Wang、A. Goldie 等，《面向机器学习编译器的可迁移图优化器》，arXiv 预印本 arXiv:2010.12438，2020年。 [29] Y. Gao、L. Chen 和 B. Li，《Spotlight：优化深度神经网络训练的设备放置》，载于国际机器学习会议。PMLR，2018年，第1676–1684页。 [30] Z. Jia、M. Zaharia 和 A. Aiken，《超越深度神经网络的数据并行与模型并行》，《机器学习与系统会议论文集》，第1卷，第1–13页，2019年。 [31] A. Paliwal、F. Gimeno、V. Nair、Y. Li、M. Lubin、P. Kohli 和 O. Vinyals，《用于优化计算图的强化遗传算法学习》，arXiv 预印本 arXiv:1905.02494，2019年。 [32] U. U. Hafeez、X. Sun、A. Gandhi 和 Z. Liu，《使用 Pesto 实现深度神经网络运算的最优放置与调度》，载于第22届国际中间件会议，2021年，第39–51页。 [33] H. Zhao、Z. Han、Z. Yang、Q. Zhang、F. Yang、L. Zhou、M. Yang、F. C. Lau、Y. Wang、Y. Xiong 等，《HiveD：在保证服务质量的情况下共享用于深度学习的 {GPU}集群》，载于第14届 {USENIX} 操作系统设计与实现研讨会（{OSDI} *20*），2020年，第515–532页。

[34] Q. Weng、W. Xiao、Y. Yu、W. Wang、C. Wang、J. He、Y. Li、L. Zhang、W. Lin 和 Y. Ding，《现实环境中的机器学习即服务：大规模异构 GPU 集群中的工作负载分析与调度》，载于第19届 USENIX 网络系统设计与实现研讨会（NSDI 22），2022年，第945–960页。[35] T. Chilimbi、Y. Suzue、J. Apacible 和 K. Kalyanaraman，《Project Adam：构建高效且可扩展的深度学习训练系统》，载于第11届 {USENIX} 操作系统设计与实现研讨会（{OSDI} *14*），2014年，第571–582页。[36] H. Zhang、Z. Zheng、S. Xu、W. Dai、Q. Ho、X. Liang、Z. Hu、J. Wei、P. Xie 和 E. P. Xing，《Poseidon：面向 {GPU} 集群上分布式深度学习的高效通信架构》，载于*2017* {USENIX} 年度技术会议（{USENIX}{ATC} *17*），2017年，第181–193页。[37] F. Seide、H. Fu、J. Droppo、G. Li 和 D. Yu，《1比特随机梯度下降及其在语音深度神经网络数据并行分布式训练中的应用》，载于国际语音通信协会第十五届年度会议，2014年。[38] N. Strom，《使用商用 GPU 云计算实现可扩展的分布式深度神经网络训练》，载于国际语音通信协会第十六届年度会议，2015年。[39] S.-C. Kao、G. Jeong 和 T. Krishna，《Confuciux：使用强化学习为深度神经网络加速器自主分配硬件资源》，载于2020年第53届 IEEE/ACM 微体系结构国际研讨会（MICRO）。电气电子工程师学会，2020年，第622–636页。[40] C. Zhang、O. Vinyals、R. Munos 和 S. Bengio，《深度强化学习中的过拟合研究》，arXiv 预印本 arXiv:1804.06893，2018年。[41] X. Song、Y. Jiang、S. Tu、Y. Du 和 B. Neyshabur，《强化学习中的观测过拟合》，arXiv 预印本 arXiv:1912.02975，2019年。[42]Sergio Guadarrama、Nathan Silberman，《TensorFlow-Slim：用于定义、训练和评估复杂模型的轻量级库》，https://github.com/google-research/tf-slim，2016年。[43] G. Karypis 和 V. Kumar，《用于划分非规则图的快速高质量多层次方案》，《SIAM 科学计算杂志》，第20卷，第1期，第359–392页，1998年。[44] Y. Bao、Y. Peng、Y. Chen 和 C. Wu，《用于加速分布式深度神经网络训练的抢占式全归约调度》，载于 IEEE INFOCOM 2020—IEEE 计算机通信会议。电气电子工程师学会，2020年。[45] D. Chen、Y. Lin、W. Li、P. Li、J. Zhou 和 X. Sun，《从拓扑视角测量并缓解图神经网络的过平滑问题》，载于 AAAI 人工智能会议论文集，第34卷，第04期，2020年，第3438–3445页。[46] C. Cai 和 Y. Wang，《关于图神经网络过平滑的注记》，arXiv 预印本 arXiv:2006.13318，2020年。[47] P. Veli˘ckovi´c、G. Cucurull、A. Casanova、A. Romero、P. Lio 和 Y. Bengio，《图注意力网络》，arXiv 预印本 arXiv:1710.10903，2017年。[48] J. Forrest、T. Ralphs、S. Vigerske、L. Hafer、B. Kristjansson、J. Fasano、E. Straver、M. Lubin、H. Santos、R. Lougee 等，《coin-or/cbc：版本 2 1317566，2018 9 9，10 5281 . .》，URL http://dx.doi.org/ . /zenodo，第 . 卷。[49] M. Wang、D. Zheng、Z. Ye、Q. Gan、M. Li、X. Song、J. Zhou、C. Ma、L. Yu、Y. Gai、T. Xiao、T. He、G. Karypis、J. Li 和 Z. Zhang，《深度图学习库：以图为中心的高性能图神经网络软件包》，arXiv 预印本 arXiv:1909.01315，2019年。[50] D. P. Kingma 和 J. Ba，《Adam：一种随机优化方法》，arXiv 预印本 arXiv:1412.6980，2014年。[51] K. Simonyan 和 A. Zisserman，《用于大规模图像识别的超深卷积网络》，arXiv 预印本 arXiv:1409 1556，2014年。[52] A. Vaswani、N. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、L. Kaiser 和 I. Polosukhin，《注意力机制即可》，arXiv 预印本 arXiv:1706.03762，2017年。[53] Y. Gao、L. Chen 和 B. Li，《Post：通过交叉熵最小化和近端策略优化实现设备放置》，《神经信息处理系统进展》，第31卷，2018年。[54] B. Jeon、L. Cai、P. Srivastava、J. Jiang、X. Ke、Y. Meng、C. Xie 和 I. Gupta，《Baechi：机器学习图的快速设备放置》，载于第11届 ACM 云计算研讨会论文集，2020年，第416–430页。[55] M. Bauer、S. Treichler、E. Slaughter 和 A. Aiken，《Legion：使用逻辑区域表达局部性与独立性》，载于 SC'12：国际高性能计算、网络、存储与分析会议论文集。电气电子工程师学会，2012年，第1–11页。