arXiv:2301.02691v1 [cs.DC] 6 Jan 2023

大模型深度学习并行与分布式训练系统

Kabir Nagrecha 加利福尼
亚大学圣迭戈分校 拉霍亚，美国
knagrech@ucsd.edu

摘要
深度学习（DL）已改变计算机视觉、自然语言处理和表格数据分析等多个领域的应用。为了提升深度学习模型的准确率，业界不断探索规模更大的神经网络架构，近期的一些Transformer 模型已拥有数千亿个可学习参数。这些设计为深度学习领域带来了由规模驱动的新系统挑战，例如内存瓶颈、运行时效率低下以及模型开发成本高昂。为解决这些问题，研究人员探索了多种技术，包括神经网络架构并行化、在内存层次结构中溢出数据，以及采用内存高效的数据表示方法。本综述将探讨大模型训练系统的发展现状，重点介绍其中的关键挑战以及应对这些挑战所采用的各种技术。

1 引言

过去几年间，深度学习（DL）已成为众多高价值应用中的关键组成部分。计算机视觉（CV）、自然语言处理（NLP）和推荐系统如今都高度依赖深度学习架构来实现预测建模与分析。深度学习的普及催生了一个新的领域——“深度学习系统”，该领域研究使用深度学习所带来的基础设施和计算挑战。

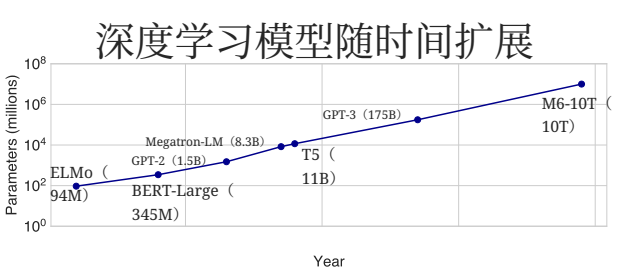


图 1：展示最先进的深度学习 Transformer 模型随时间增长的情况。纵轴（模型参数数量）采用对数刻度。

深度学习实践的最新进展为深度学习系统研究带来了一个迫切的新挑战：模型规模 。实践者已开始探索使用超大型神经网络架构图来构建深度学习模型，其中一些模型包含数十亿甚至万亿个可训练参数！典型例子包括 NLP Transformer模型 BERT-Large [16], GPT-3 [13] 以及 Meta 的深度学习推荐模型（DLRM） [41]。这些模型的庞大规模在三个关键领域带来了严峻挑战。

(1) 内存可扩展性。 标准的深度学习训练通常将模型参数保存在加速器（例如图形处理器）的内存中，并使用采样数据为每个参数计算梯度更新。对于非常大的模型，存储参数、中间计算和梯度更新所需的空间通常会超出加速器相对有限的内存容量。高端消费级图形处理器（如 Tesla V100 [2] ）的设备端内存为 16–32GB，但大型 DLRM 可能需要数百 GB 的空间。

(2) 性能。 参数数量的增加通常与更长的执行时间相关。此外，复杂的模型架构往往需要大型数据集才能充分发挥模型的学习能力——例如，GPT-3 使用 300B 词元进行训练 [13] ，开源 BLOOM 语言模型则使用 366B 词元进行训练 [12]。训练此类模型可能需要数周甚至数月 [12, 41]。因此，能够提升执行性能的优化对于大规模深度学习模型的开发者而言大有裨益。

(3) 训练成本。 针对前两项挑战的标准解决方案通常涉及在多个设备之间并行化存储或执行。然而，这种方法会大幅推高计算成本。BLOOM 使用 416 个 A100图形处理器进行训练，而 Megatron-LM 使用了 512 [5]。对于大多数从业者而言，这并不现实，尤其是这些图形处理器需要预留数周甚至数月的训练时间。在 AWS 上复现 BLOOM 的训练流程将耗资 550 万美元。请注意，这还未计入模型选择的额外成本；在模型选择过程中，需要训练多个模型，以评估最佳的超参数设置和配置 [29]。

随着从业者不断突破模型规模的边界，为了推动大模型深度学习领域的进一步发展，解决这些挑战变得愈发必要。因此，人们开发了各种系统和技术，分别应对其中的一个或多个问题。一些方向包括重计算 [15], 数据溢出/CPU卸载 [23, 36, 37, 45–47],流水线/模型并行 [21, 25, 27, 33, 40], 以及混合并行 [26, 31, 36, 37, 60]。这些主题统称为“大模型训练技术”，已成为业界和学术界研究人员关注的重点，但该领域的研究成果数量庞大，令这一主题难以梳理。本文将全面回顾当前大模型深度学习训练系统的发展现状，并对

该领域未来的发展方向进行评估。

近期已有一些关于该领域的高层次简短综述问世 [20, 54]，但这些综述并不全面，也未涉及模型选择、混合并行以及各种技术交叉应用等关键主题。本文将探讨这些领域，并进一步深入介绍最先进的技术。

请注意，本综述不涵盖图神经网络训练，因为这类架构所面临的问题与其他深度学习模型有很大不同，需要另行撰写一篇综述进行深入探讨 [6]。本文也不涉及“模型缩减”技术（例如模型蒸馏），因为这些技术超出了训练的范畴，属于模型修改 [18, 44, 57]。为求简洁，本文也不涵盖混合专家模型，因为它们引入了不同且复杂的性能模型。

本文结构如下：第 2 节介绍一些必要的背景知识和术语。第 3 节描述大模型架构及一些关键示例。第 4 节探讨大模型深度学习训练系统的全貌。最后，第 5 节总结本综述，并探讨未来可能的发展方向和新兴挑战。

2 背景与预备知识

首先介绍本文后续内容所需的一些关键技术概念和符号。

2.1 深度学习

深度学习（DL）是一种通过对有向算子图的参数进行梯度下降优化，来近似数据中模式的技术。在本文中，我们将 DL 模型图称为模型或神经计算图。前馈网络是一种特殊的计算图，其中输入沿着算子链持续向前传递，每个算子称为一层。大多数大规模模型架构都属于这一类。

常见的深度学习算子包括矩阵乘法、可微激活函数和嵌入表查找（在功能上等同于与独热向量进行矩阵乘法）。前两者用于直接变换数据输入，而后者用于将类别型输入映射为某个向量（例如，将应用程序用户转换为表示该用户在潜在空间中位置的向量）。深度学习模型可能会将所有这些算子组合成复杂结构，这种结构通常称为深度神经网络（DNN）。

模型的训练 涉及一种称为随机梯度下降（SGD）的优化过程。它从训练数据集中采样小批量（互不重叠的子集）数据，然后在模型计算图中执行前向 传播。前向传播将输入数据特征转换为输出目标，然后根据某些真实标签计算损失值或误差。该损失值用于执行反向 传播，也称为反向传播。反向传播是指反复应用导数链式法则，为每个模型参数计算基于梯度的更新，从而最小化输出损失的过程。请注意，要使这一链式过程有效，算子之间的中间输出（也称为激活值）必须保存。推理在这些应用中，模型参数被冻结并部署用于预测；转换输入时可以丢弃中间输出，但训练时必须保存中间结果（这可能会大幅增加内存成本）。

将小批量数据遍历整个数据集一遍（即抽取足够多的子集，使模型看到全部数据）称为一个训练轮次。训练模型通常需要进行多个训练轮次的随机梯度下降。常见的深度学习工具实现了许多随机梯度下降的变体（例如 Adam、AdaGrad、RMSProp 等），这些变体称为优化器 ，但它们的数据访问模式相同。有些优化器会维护自己的参数和状态，并在执行过程中更新它们

深度学习模型的大小可以通过参数数量和/或内存占用来衡量。模型的内存占用通常与参数数量相关。为了完成一次完整的小批量数据遍历，必须在各层之间存储中间数据；在反向传播过程中，梯度沿链路反向传递时也需要保存在内存中。因此，可以根据模型参数、中间数据和梯度各自的内存成本总和，估算模型内存占用的上限。这个上限通常会被大幅高估，因为大多数训练框架会在中间数据和梯度不再需要时自动将其丢弃。

2.2 深度学习加速器

矩阵乘法通常是深度学习模型计算图中最常见且计算密集度最高的算子。其他算子也存在（例如激活函数、嵌入表查找），但其计算密集度往往较低。通用矩阵乘法运算，即 GEMM，是一个经过充分研究、适合并行化的目标 [24]。因此，使用能够有效利用这一机会的硬件，是深度学习模型开发者的常见做法。

图形处理器 相比中央处理器，在执行 GEMM 等简单运算时具备更强的并行化能力。具体而言，英伟达图形处理器支持 cuDNN 库 [1],，该库实现了针对图形处理器线程并行化进行优化的底层深度学习算子。然而，图形处理器必须处理存储在加速器内存中的数据，而加速器内存的容量远小于标准系统内存（动态随机存取存储器）。目前先进的消费级图形处理器通常只有不到 40GB 的设备端内存，而训练机器中的动态随机存取存储器容量则可能轻松超过 512GB。

TPU（即张量处理单元）是谷歌专为深度学习开发的一种新型加速器。TPU 核心的组织方式支持通用矩阵乘法运算中的并行，实验评估表明，在某些模型架构上，其性能最高可比图形处理器快 10 倍 [58]。遗憾的是，目前 TPU 的使用权限仍然十分有限——从业者只能通过谷歌云平台使用 TPU。因此，本综述将主要关注面向图形处理器执行的系统。

2.3 并行化技术

存在多种用于并行化深度学习训练的技术。每种技术各有优缺点，具体表现往往取决于工作负载。要在大模型训练场景下评估这些策略，理解每种策略的数据访问和通信模式至关重要。

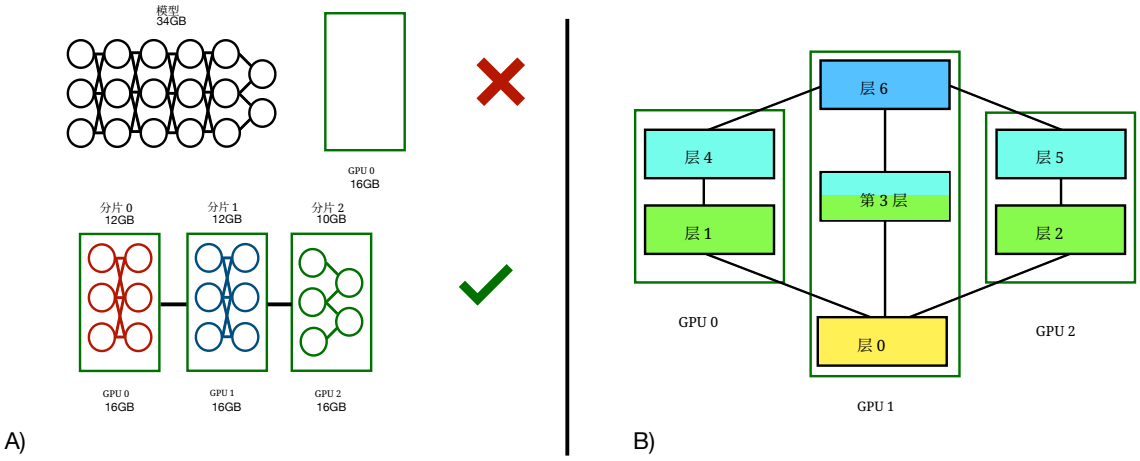


图 2：A）示意如何将无法装入单个图形处理器的大型前馈网络，拆分到三个图形处理器上进行模型并行，以便执行。请注意，执行速度并未提升——这并非并行执行，只是将内存需求分散到了不同设备上。B）示意一种在三个图形处理器上高效的模型并行分片策略。用相同颜色标示的是可并行执行的层。该策略利用算子计算图中现有的并行执行机会——如果计算图的结构更加顺序化，这种做法不一定可行。

模型并行是指将神经网络架构图划分为多个子图（即分片），并将每个子图（或模型分片）分配给不同设备的技术。对于前馈网络，这些分片可能对应于一组堆叠层。

模型并行的加速潜力很大程度上取决于架构和分片策略。对于前馈网络，像图 2A）所示那样按顺序进行模型分片，并不能实现并行执行，反而会在加速器之间形成依赖图。不过，对于顺序式架构（例如 Transformer），这种分片策略仍然很常见，因为它能够将内存需求分散到多个设备上，而且设置相当简单。在其他情况下，神经计算图则会自然地提供算子间并行的机会。图 2B）对此进行了示意。

另一种方法是直接对网络中的各个算子进行分区。某些算子（例如嵌入表）可以按宽度分片，且开销极小。另一些算子（例如矩阵乘法）也可以进行分区（例如采用并行通用矩阵乘法策略 [55]），但会涉及更多通信步骤。图 3 展示了模型并行化嵌入表的示例。这些按宽度分片策略通常也称为张量并行，因为它们需要对输入张量进行分区。与算子间模型并行相比，这种策略可以实现性能更高的算子内并行，但实现起来需要更多投入和考量。此外，由于大多数张量并行算子都至少需要一次全收集通信步骤来重新聚合分区后的输出，因此其性能优势会受到限制。Mesh-TensorFlow [51] 提供了一个通用框架和语言，用于指定张量并行计算；不过需要注意的是，它已不再得到积极支持或维护。

任何形式的模型并行都会引入图形处理器之间的通信。英伟达最新的图形处理器支持“NVLink”互连——这是一种高速的图形处理器间通信通道，带宽最高可达 900GB/s，有助于尽量减少开销。不过，NVLink 并非总是随时可用，尤其是在使用用户难以自行定制的云端提供的机器时。如果不支持 NVLink，图形处理器之间的通信就会通过 PCIe 互连进行，速度要慢得多。Tesla V100 通常被视为深度学习应用的标准高性能图形处理器，它支持 16 通道的 PCIe 3.0 互连，带宽为 16GB/s。

为了避免通过速度较慢的互连传输过多数据，模型并行用户通常会努力选择一种分区策略，以尽量减少需要在分片之间传输的激活值大小，或者平衡计算负载以隐藏通信成本。对此存在多种分片算法 [14, 25, 26, 37, 46, 61]。

数据并行是一种常见的深度学习执行策略，可让多个小批量数据并行处理。数据并行执行技术可分为两大类：异步数据并行和同步数据并行。

最知名的异步技术是参数服务器：一台核心主服务器持有一组基准参数，而分布式工作节点则持有模型副本，并使用不同的小批量进行训练。分布式工作节点会偶尔向基准服务器发送更新，基准服务器随后会将替换参数发送给这些工作节点，以使其保持更新。工作节点之间可能会出现不同步，因为它们只需与基准服务器通信／同步。异步技术会带来许多挑战，例如与单工作节点训练相比准确性下降，以及由于工作节点返回时间存在差异而导致结果不可复现。因此，在现代深度学习训练领域，异步技术通常正逐渐被同步技术取代。

最常见的同步数据并行执行技术是分布式数据并行（DDP）。DDP 会复制模型，并

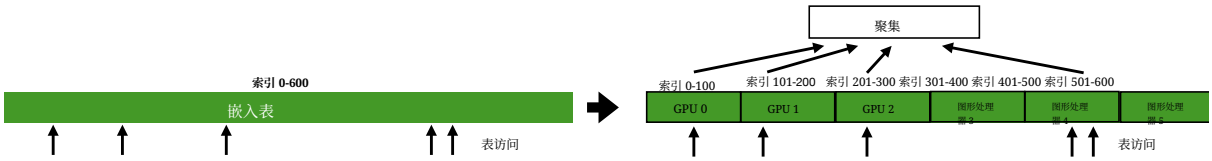


图 3：在 6 个图形处理器上并行化的嵌入表。每个图形处理器接收表中不同子集的索引。

将副本分配到 m 不同的加速器上。首先输入一个“全局小批量”，然后将其均匀划分到各个副本上，为每个副本生成本地梯度更新。随后，在各副本之间聚合这些梯度，生成一次全局更新，通常采用全归约通信模式。接着，所有副本并行应用这次全局更新。这种技术在数学上等同于使用原始全局小批量进行单个图形处理器训练。虽然这种技术会产生全归约通信步骤，但通常可以将这些开销与模型执行时间重叠并隐藏起来。

全收集和全归约通信模式常用于数据并行和更广泛的分布式深度学习。这些模式旨在将不同处理器上的独立数据聚合起来，再分发回各处理器，使每个处理器都持有相同数据的副本。全收集模式采用一种算法，让每个处理器将自己的数据发送给其他所有处理器。通常，当每个处理器持有一部分需要全局广播的数据时，会使用这种模式。这通常要求每个处理器执行 n 个通信步骤，并消耗大量带宽 —— 每个处理器都必须与其他所有处理器通信。全归约模式是在全收集之上增加一层，将聚合与某种归约函数（例如求和、求平均）结合起来。在同步期间运行该函数，相比先执行全收集、再在本地应用函数，可以省去一个步骤。例如，Horovod [50], 为数据并行梯度聚合实现了一种带宽最优的归约模式 [43]，每个处理器只需执行 $2 \times (n-1)$ 个通信步骤，即可完成整个聚集与归约过程。

混合并行是指结合不同并行化策略以提升整体性能的策略。例如，在模型并行之上叠加数据并行，可以使用户在多个设备上实现内存可扩展性，同时获得数据并行带来的执行加速。这些策略各有取舍，设计时需要将其纳入考量。在简单的叠加式混合并行中，模型并行对多设备的需求会乘以复制所需的

数据并行的实现。进一步叠加任务并行（例如多模型训练），还可在等式中引入另一个乘数因子。更复杂的混合方式可以在模型并行架构的某些阶段应用数据并行，而让其他阶段串行执行，如图 4 所示。请注意，这种设计开辟了新的“搜索空间”——应选择哪些阶段进行数据并行复制？模型并行分片决策会如何影响数据并行性能？应如何将有限资源分配给各个阶段？设备互连和拓扑结构又会如何影响性能？第 4.4.2 节介绍了用于回答这些问题的各种混合并行策略选择工具。第 4.4.1 节则描述了另一类混合并行设计，它们尝试提出新的形式，将基本的模型并行、数据并行和任务并行方法相结合，实现并行执行。

尝试提出新的形式的并行执行方式，将基本的模型并行、数据并行和任务并行方法相结合，相关内容见第 4.4.1 节。

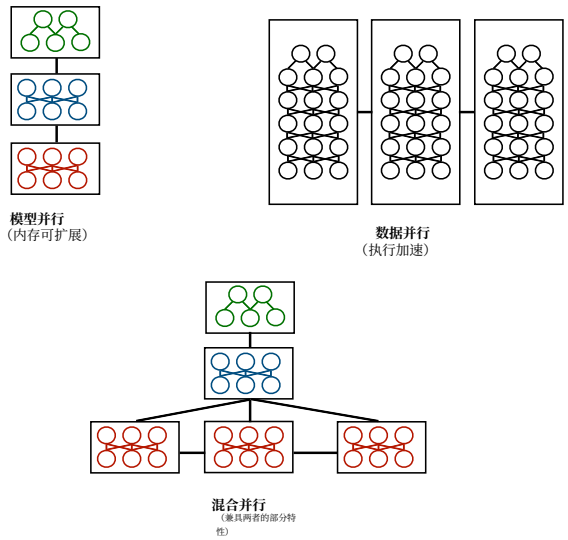


图 4：模型并行、数据并行和混合并行的对比示意图。模型并行使大模型训练能够利用多个图形处理器的内存，但其加速潜力在很大程度上取决于架构。数据并行为大多数架构提供了一种通用且简单的执行加速方式。混合并行则提供了结合两者的方法，但也带来了新的挑战，包括如何找到最优的混合策略和资源分配方案。

3 大模型架构 S

为阐明大型模型训练系统的必要性并推动其设计，我们介绍各种大模型架构及其各自带来的独特挑战。模型架构的规模大致可分为两类——深度扩展和宽度扩展。对于 Transformer 等长序列链式架构，通常需要进行深度扩展。宽度扩展则常用于非常宽、易于并行化的算子（例如表查找）。第三种情形是样本扩展，指输入样本的大小而非模型大小带来扩展挑战的情况。这种情形不属于“大模型训练”的范畴，因此本文不对此进行深入讨论。

3.1 深度模型与 Transformer

Transformer是最常见的深层模型架构示例，近年来已在众多领域广受欢迎。自然语言处理（NLP）、计算机视觉（CV）乃至表格数据分析中的基准任务，如今都由Transformer架构领衔。这些模型由多个堆叠的“自注意力”模块组成，每个模块又包含多次点积运算和矩阵运算。实践者发现，堆叠更多注意力模块以增加Transformer的深度，通常能提升准确率 [4]。因此，Transformer促使实践者不断探索更深的模型架构。

这些模型给训练带来了严峻挑战。早期的大型 Transformer 之一是 BERT-Large [16], , 使用了 3.45 亿个参数。要在图形处理器上以合理的小批量大小训练这个模型，所需内存就已经迫使从业者使用高端图形处理器。此后，Transformer 的深度不断增加——近期的一些模型已达到一万亿个参数，所需空间远远超出了市面上任何 图形处理器的内存容量。

因此，对于大型 Transformer 训练以及一般的深度模型训练而言，模型并行等技术实际上必不可少。然而，要让极深模型实现并行执行，可能颇具挑战。对于由多层组成的深层序列，最自然的分片策略是将序列划分为多个子序列。但这种方法迫使用户增加图形处理器，却无法从中获得任何性能提升——将序列划分为多个子序列并不能带来并行执行加速的机会。试想一个需要使用 1024 个图形处理器才能装入内存的一万亿参数模型。这些图形处理器仅仅是为了“启用”执行而被使用，并未带来任何性能提升。事实上，由于 GPU间通信成本，这种策略很可能比同等的内存内训练作业更慢。

目前存在一些按宽度分片的策略，例如在多个图形处理器上并行执行注意力模块中的操作。然而，这些方法需要进行更多定制，会增加通信开销，并且需要模型设计者投入大量精力来实现。因此，大多数深度模型训练系统更倾向于采用一种可针对所有深度模型类别进行优化的通用按深度分片策略，而不是每次只针对单一架构。

尽管存在顺序依赖这一挑战，按深度分片也能带来许多机会。诸如 流水线并行 和 溢出 之类的技术只能用于按深度分片的模型。我们将在第 4 节进一步介绍这些技术。

3.2 宽模型与嵌入表

宽模型面临的挑战与深模型截然不同。虽然以高效发挥性能的方式对宽模型进行并行化更容易（按宽度而非按深度），但按宽度分区通常需要执行全收集通信步骤，以聚合并行化后的部分输出。

嵌入表 通常是推荐模型中最适合按宽度分片的对象。基于嵌入的推荐模型被大多数收集实体特定数据的公司（例如 Meta、Netflix、TikTok）所采用，以打造个性化体验。常见做法是创建一张将用户ID映射到可训练向量的表，再将这些向量输入上层的某个深度神经网络。然而，要让这种方法适用于 Facebook 这样拥有数十亿用户的平台，相应的表就必须非常宽。一张包含三十亿个索引、每个索引对应 1024 个由单精度（32 位）浮点数组成的可训练向量的表，需要 12TB 内存。实际的推荐系统可能会针对不同的查询使用多张 此类表（例如用户表、商家表、视频目录表），从而进一步增加内存成本。

对嵌入表进行分区是一项简单的任务，因为查表具有令人难以置信的并行性——对一个索引的查找不依赖于其他索引。因此，将一张表分片为分配给不同图形处理器的子表，是分摊内存成本的常见策略。只需将小批量中的索引查找请求路由到相应的图形处理器，即可轻松实现跨分片的并行执行。然而，为了在并行查表之后重新聚合小批量，以便将其输入顶部深度神经网络，必须执行一个可能开销高昂的全收集通信步骤。

虽然不太常见，但也并非闻所未闻：对矩阵乘法等其他算子采用按宽度分片。不过一般来说，嵌入表是内存占用最大的单个算子 [9]。鉴于按宽度分片的主要应用场景是嵌入表，针对这一场景进行优化似乎过于狭隘。然而，嵌入表和推荐模型在深度学习工作负载中占比远高于一般水平——Meta 报告称，其 50% 的深度学习训练周期都用于基于嵌入表的推荐模型 [9]。因此，即使其适用范围比针对顺序深度模型可扩展性的优化更有限，优化超宽模型场景也完全值得投入。

4 大型模型训练系统

为应对如何实现高效大模型训练这一挑战，已经涌现出若干系统。下面介绍大模型训练系统的主要类别，以及各类别中的关键实例。图 5 以表格形式对本综述涵盖的各类系统进行了比较。

4.1 基本技术

一些基本技术（如重计算）常被用作更先进的大模型训练系统的通用构建模块。总体而言，这些技术对组织和结构的影响很小，因此易于与其他方法结合。

4.1.1 重计算。重计算也称为梯度检查点，旨在尽可能降低反向传播的内存需求 [15, 19]。如第 2 节所述，反向传播需要保存中间算子的输出，以便正确应用用于梯度计算的链式法则。然而，中间输出张量可能会占用大量内存！一些分析 [54] 表明，激活值最多可占 ResNet [22] 内存消耗的 95%，并且

	技术名称	No 精度下降	加快计算速度				增加模型可扩展性
			数据并行加速	任务并行加速	模型并行加速	直接执行加速	
Basic Techniques	检查点	✓	✗	✗	✗	✗	✓
	累积	✓	✗	✗	✗	✗	✓
	低精度表示	✗	✗	✗	✗	✓	✓
	稀疏表示	✓	✗	✗	✗	✗	✓
	模型并行	✓	✗	✗	✓	✗	✓
	数据并行	✓	✓	✗	✗	✗	✗
	张量并行	✓	✓	✗	✓	✗	✓
Pipeline Parallelism	GPipe	✓	✓	✗	✓	✗	✓
	DAPPLE	✓	✓	✗	✓	✗	✓
	PipeDream	✗	✓	✗	✓	✗	✓
Spilling	SwapAdvisor	✓	✗	✗	✗	✗	✓
	L2L	✓	✗	✗	✗	✗	✓
Hybrid Parallelism	ZeRO	✓	✓	✗	✓	✗	✓
	Hydra	✓	✗	✓	✗	✗	✓
	Megatron	✓	✓	✗	✓	✗	✓
	FlexFlow	✓	✓	✗	✓	✗	✓
	Alpa	✓	✓	✗	✓	✗	✓
	DLRM MP-DP	✓	✓	✗	✓	✗	✓

图 5：各种大型模型训练系统和技术的完整比较。

在某些 Transformer 中，检查点可能占用高达 80% 的内存。重新物化通过以计算换取内存：首先丢弃大部分激活值，仅保留少数检查点，然后在反向传播期间利用这些检查点重新计算被丢弃的内容。这样，在任意时刻只需将检查点之间的中间结果存储在内存中。这种方法确实会带来计算开销——前向传播实际上要运行两次。不过，前向传播中的算子通常比反向传播中使用的自动微分过程快，因此开销没有看起来那么大。一些梯度检查点系统声称，只需 30% 的开销即可节省 6 到 7 倍的内存 [15]。检查点对于流水线并行和分片交替并行等技术至关重要，相关内容将在第 4.2 节和第 4.4.1 节介绍。

4.1.2 累积。累积旨在降低反向传播中批量梯度的内存需求 [25]。第 2 节介绍了随机梯度下降如何将样本分组成小批量，并将其输入模型。相应地，我们可以将用于参数更新的梯度视为对每个样本分别执行更新后再进行汇总的结果。累积会延迟应用这些汇总后的梯度，转而计算新的小批量梯度更新，并将其累加到汇总梯度向量中。此时的新梯度是*2* 个小批量更新的汇总，而非 1 个。这样，我们无需实际训练更大的批量，就能扩大有效小批量大小以及梯度的影响。我们将较小的单独批次称为微批次，而有效汇总后的批次仍称为小批量。累积对于流水线并行至关重要，而且常与其他技术结合使用。

4.1.3 低精度表示。大多数训练框架（例如 TensorFlow、PyTorch）[8, 42] 使用单精度浮点数（32 位）表示梯度和参数。双精度表示（64 位）相对少见。降低模型训练内存需求的一种方法是使用半精度（16 位）表示数据。自然，这会因数值被近似处理而造成精度损失 [38]。不过，这种方法可以同时提升速度并节省内存。为尽可能兼顾两者，自动混合精度（AMP） [3] 会自动尝试判断哪些数据可以安全地压缩为 16 位，而不会造成精度损失。AMP 通常几乎不会造成精度损失，同时在训练大型模型时最高可实现 5.5 倍加速 [3]。由于 AMP 直接在极低层级修改数值，因此这项技术通常与大模型训练的实际系统方法相互独立。

4.1.4 稀疏表示。在某些情况下，深度学习训练中使用的向量非常稀疏。例如，嵌入表查找通常只涉及表中的少数索引。应用于该表的梯度向量只会在使用到的索引处具有非零值，其余梯度值则会被置零。实际上，在内存中保留所有这些零值并无必要，只会浪费内存。稀疏表示旨在将这些向量压缩为非零值，同时避免信息丢失。最简单的方法通常是嵌入表的默认表示方式，即将梯度表示为键值对将索引映射到梯度值。如下图所示。

$$< 0{,}0,\ 0,\ 0,\ 0,\ 0,\ 4,\ 0 > \rightarrow \{6:4\}$$

这种表示形式可以轻松映射回去以供应用，同时丢弃不必要的零值。

将稀疏表示与假设采用标准向量表示的操作（例如全归约通信模式）结合时，会产生一些复杂问题。一些研究 [35] 展示了如何通过采用替代通信模式或将数据转换回标准表示来解决这些问题。稀疏向量表示针对的是一个非常具体的问题，但对于某些算子（例如宽嵌入表）的高效训练至关重要。

4.2 流水线并行

流水线并行针对的是“顺序深度模型”场景 [25]。它是第 2 节所述模型并行训练范式的直接扩展。模型并行会创建一个分阶段的分片序列，从而形成自然的“管道”结构。流水线并行只需利用这种管道结构，尝试让各阶段填满执行操作，以减少纯顺序模型并行中存在的空闲时间。以一个纯前馈网络为例，如图 2A 所示。每个分片都可以视为管道的一个阶段，因此，将模型划分为三部分并分布在三个图形处理器上的方式，现在构成了一个三阶段流水线。

4.2.1 流水线并行基础。 在 CPU 流水线中，我们将各种指令发送到中央处理器，以填满流水线[52]。对于深度学习流水线并行，我们用微批次填满流水线，这些微批次与梯度累积中使用的微批次类似 [25, 27]。本质上，流水线并行是梯度累积与模型并行的结合。相互独立的微批次在分片管道中依次传递，然后在每个流水线阶段累积各微批次的梯度。整个小批量（由所有微批次组合而成）的梯度全部聚合后，就可以将其应用于模型。这种设计近似于模型并行与数据并行的混合方案：数据分片并行处理，但使用不同的模型并行分片。图 6 展示了这一过程。

4.2.2 挑战。 反向传播给流水线并行训练带来了挑战。如第 2 节所述，必须保留中间输出才能进行反向传播。然而，与累积结合时，我们就需要为每个微批次存储一组不同的中间输出，从而失去累积所带来的任何可扩展性优势。GPipe [25], 是最早的流水线并行训练系统之一，它提出将累积与检查点结合起来解决这一问题。激活值只在分片／管道阶段的边界处存储；在反向传播期间，随着梯度沿管道向后传递，再重新计算激活值。如今，大多数（如果不是全部）流水线并行训练系统都采用检查点方法 [21]。

管道的结构也带来了另一项挑战。要实现流水线并行，分片管道必须是双向的。预测期间，输入和激活值向前流动；反向传播期间，梯度则向后流动。这会导致一个问题——数据

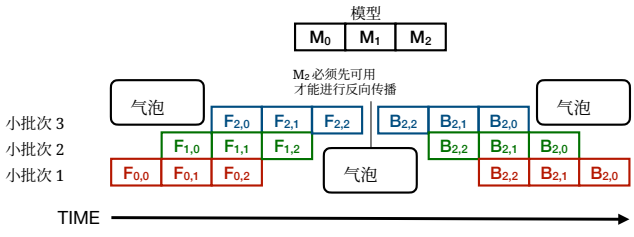


图 6：三个分片上的流水线并行示意图。输入小批量被划分为微批次，随后在各流水线阶段之间传递。$F_{x,y}$表示分片 x 上针对小批量 y的前向阶段，而 $B_{x,y}$ 表示分片 x上针对小批量 y 的反向阶段。通过这种方式，实现了一种“流水线式”的数据并行，即数据在不同模型并行阶段之间**并行处理。请注意，在反向传播之前，必须清空前向流水线。**

在管道中双向流动时，会在各个阶段“碰撞”。因此，在预测和反向传播之间会发生流水线清空。若管理不当，清空操作会严重影响性能。图 6 展示了一个采用流水线并行的模型。请注意，大量时间都耗费在“气泡”阶段，此时必须彻底清空流水线。

4.2.3 主要流水线并行系统与技术。人们已尝试多种方法来应对上述挑战。GPipe [25] 提出在加速器数量不变的情况下增加微批次数量，使管道能够更长时间保持满载。这无法消除清空操作，但可以提高整体效率。然而，这种方法需要更多内存来存储更多检查点化的微批次激活值。DAP-PLE [17] 提出了一种替代的流水线调度，在保持 GPipe 收敛行为的同时减少空闲时间。遗憾的是，由于同时保持“活跃”状态的微批次增多，这种方法也大幅增加了内存成本，因此对于已经面临内存极限的应用而言，这种调度并不可行。

另一种解决方案是异步流水线，它通过重新排列流水线阶段和反向传播来消除清空流水线的过程，但代价是必须维持严格的收敛行为。这种对顺序的“解耦”将问题放宽为效率更高的问题，却会影响数据消费顺序和消费量 [21, 32, 39, 59]。例如，PipeDream 提出的 1F1B 模式 [21] 会针对每个反向阶段运行一个前向阶段（处理不同的微批次），以维持完美的比例和利用率。但其设计需要更加谨慎地进行分区和打包；要减轻过时权重更新导致的精度下降，就必须暂存权重的多个副本 [21],，从而大幅增加内存成本。虽然 1F1B 等异步流水线可以有出色表现，但它并非通用解决方案——精度损失因具体情况而异，而且往往相当显著。对于精度至关重要且必须能够复现收敛行为的应用（例如模型选择），异步流水线并不适用。

4.3 内存卸载与溢出

模型并行着眼于通过多个图形处理器上的执行来分摊内存需求，而有些系统则尝试利用主系统内存（动态随机存取存储器），而不是通过增加图形处理器来横向扩展。采用这种方法的主要原因是，GPU 内存容量有限且价格昂贵，而动态随机存取存储器则便宜得多，也更易于获取。

4.3.1 动机。以 AWS 提供的 p3.2xlarge 节点为例。它配备一块 V100 GPU，拥有 16GB 设备端内存。大型深度学习模型可能无法装入该节点的 GPU 内存，但实际上，该节点仍有更多可用总内存——61GB 动态随机存取存储器，加上 16GB GPU 内存。在机器学习系统文献中，动态随机存取存储器有时也称为 CPU 内存 [1]，它远比 GPU 内存便宜，而且供应量也更充足；标准的云端多 GPU 机器通常轻易就能配备数百 GB 的动态随机存取存储器。

如果能将模型的内存需求分散到动态随机存取存储器和 GPU 内存中，就可以训练大型模型，而无需承担模型并行所需的多 GPU 成本。按需使用的 AWS p3.2xlarge 节点可提供 77GB 的总内存，每小时收费 3.06 美元。若要获得相同容量的纯 GPU 内存，则需要一台配备 8 块 GPU 的 p3.16xlarge 节点，用户每小时需支付 24.48 美元，成本是前者的 *8* 倍。成本优势显而易见，因此，将模型的部分内存需求卸载到动态随机存取存储器，而不是扩展到多块 GPU，对于 GPU 访问权限有限且注重成本的用户（例如小型企业和研究人员）来说，是一种很有吸引力的选择。

4.3.2 卸载系统与技术。许多早期工作 [23, 30,34, 49, 56] 将卸载视为“交换”问题——决定何时将张量从 GPU 内存换出到动态随机存取存储器。大多数工作使用图分析算法，根据激活值、梯度或参数在执行图中下一次可能被使用的时间，确定应在何处“插入”交换操作。SwapAdvisor 是这些交换系统中最先进的一个，它使用并行遗传搜索算法分析交换算子的最佳放置位置，以实现最佳性能。它也是最早支持卸载参数 以及激活值的系统之一，这对训练十亿参数模型架构至关重要。

这些复杂的交换流程可能很难设置——SwapAdvisor 的搜索算法大约需要一小时才能完成。此外，这些流程难以扩展到多 GPU 训练，因为尚无明确的方法将注入交换操作的计算图技术扩展到多 GPU 并行场景。

另一种方法是提出 ZeRO-R [46],，这是一种卸载系统，可动态地将激活值和参数发送到动态随机存取存储器。该方法采用“按需卸载”，而不是预先规划卸载。设计的不规则性可能会带来内存碎片等问题，但与基于图的设计相比，它具有更大的灵活性。后续版本 ZeRO-Infinity [47] 将这一方法扩展到非易失性内存 express/磁盘存储卸载，以进一步提升可扩展性。

Hydra [37] 选择“独立块”策略，将模型架构划分为子模型（类似于模型并行），这些子模型

[1]我们认为，“CPU 内存”一词含义模糊，因为它可能被理解为寄存器或 CPU 缓存。本文使用“动态随机存取存储器”指代主系统内存，但需要注意，其他机器学习系统论文可能会用“CPU 内存”这一说法指代同一概念。

随后可以在动态随机存取存储器和 GPU 内存之间自由溢出。可以将其类比为关系数据库管理系统中的溢出，即独立的数据块可以被发送到较低层级的内存中。与其他溢出系统不同，Hydra 的执行模式与模型并行完全相同，并将每个模型分片的执行彻底分离。它仍会尝试让通信与计算重叠，但忽略了其他 CPU卸载技术所探讨的细粒度张量卸载的复杂性。这种泛化使其在单图形处理器执行时并非最优，但更适合与多GPU并行化技术结合。我们将在第 4.4.1 节进一步展开说明。

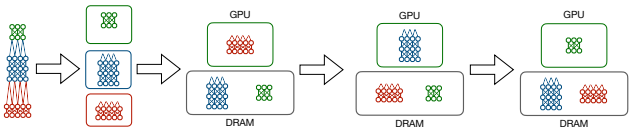


图 7：Hydra 的溢出策略很简单，就是将模型并行分片提升至 GPU 内存中，或从 GPU 内存中降级移出。其他溢出设计（例如 ZeRO-Offload 所采用的设计）与此类似，但结构上的限制较少。

L2L [45] 采用了与 Hydra 类似的设计，但其分片方法限制更多。它专门针对 Transformer架构，并以专为其目标模型类别选择的启发式方法，将自注意力块（标准 Transformer 算子）进行替换。这使其在 Transformer架构上表现出色，但无法像 Hydra 那样灵活，也不具备 ZeRO-R 的动态通用性。

请注意，这些技术通常用于沿深度方向分散大型模型的内存需求，因为它们都利用了执行过程中的某种顺序关系。无法在不显著降低性能的情况下序列化的超宽算子（例如嵌入表），很难在动态随机存取存储器和 GPU 内存之间进行溢出。对于宽算子，混合设备执行的唯一选择是：要么序列化并行算子（在表的场景中即索引查找），将一系列操作改写为深层而非宽层模型；要么直接在中央处理器上执行宽算子。

4.3.3 卸载计算。有些系统更进一步，直接在中央处理器上执行操作。一般而言，最好完全使用图形处理器或张量处理单元进行计算，因为大多数深度学习算子在支持高度并行的加速器上运行得快得多。不过，采用卸载时，数据反正会在中央处理器上，因此，让中央处理器操作与图形处理器操作并行执行应该不会增加开销。

ZeRO [48] 提出在 GPU 执行期间，使用中央处理器对参数进行更新，具体针对广受欢迎的 Adam优化器 [28]。Adam优化器会保存一些状态参数（通常为 32 位），并且需要在 32 位参数上运行，以避免精度下降。遗憾的是，这使用户无法利用 16 位表示来减少内存需求。ZeRO 版本的 Adam优化器会在动态随机存取存储器上保留 32 位参数版本，并在图形处理器上保留低精度的 16 位版本，以减少内存占用。执行期间，系统会将梯度和优化器状态转存到动态随机存取存储器中，然后对参数进行更新，使用 中央处理器处理 32 位参数。更新会在第二个步骤中传播回 16 位参数，同时使 CPU-GPU 通信与 GPU 计算重叠。

对于超大型推荐模型，CPU-GPU 混合计算也很常见。如第 3.2 节所述，嵌入表是非常宽、内存密集型的算子，通常会将结果传入规模较小的深度神经网络进行进一步处理。若不进行任何优化，嵌入表的庞大规模将迫使系统仅 CPU 执行 [9]。或者，用户可以将嵌入表放在中央处理器上，同时让深度神经网络驻留在图形处理器的内存中，以享受 GPU 加速带来的优势。一些研究（例如 Hotline [10]）尝试让数据通过流水线流经模型，从基于中央处理器的嵌入表进入 GPU 加速的深度神经网络。他们证明，这种混合计算方式甚至可以比按宽度划分的多图形处理器模型并行更快，因为它消除了第 3.2 节所述全互连通信步骤的需求。

4.4 混合并行

第 2.3 节中介绍的基本并行化技术可以通过不同方式组合。各种系统都尝试结合多种“基本”并行执行方法（例如数据并行、模型并行）的优势，为用户提供更高的性能和可扩展性。混合并行技术可分为两大类——从底层整合并行化技术的“真正”混合并行，以及在执行的不同阶段选择不同策略的自顶向下混合并行。

4.4.1 自底向上混合并行。传统上，将模型并行与其他技术从底层结合起来一直颇具挑战。模型并行会增加标准执行所需的图形处理器数量，因此与基于复制的并行技术或多实例技术（例如数据并行、任务并行）结合时可能并不实用，因为这些技术会进一步增加模型并行对设备的需求。

为了解决这一问题，Hydra [37] 提出采用第 4.3 节所述的溢出技术，以减少可扩展模型并行训练所需的图形处理器数量，然后在此基础上叠加一层任务并行，从而支持高效的多模型训练。随后，Hydra 系统利用模型并行的分段特性，实现混合的“细粒度并行”调度，其性能可超越标准任务并行和模型并行。图 9 展示了这一点。目前，Hydra 是唯一明确针对超大型模型的多模型场景设计的系统；但随着从业者日益难以承担模型选择和多用户集群管理的成本，这一领域的重要性很可能会不断提升。

全分片数据并行（FSDP）最初随 ZeRO [46], 推出，是模型并行与数据并行的混合方案。与 Hydra 不同，Hydra 仍以模型并行分片的方式执行；FSDP 仅使用模型并行来分布模型，使其分布在各个数据并行实例上，每个数据并行副本都持有某个层组的部分参数。执行层组时，FSDP 会执行全收集步骤，在每个数据并行实例上生成完整的未分片层组。随后，层组以纯数据并行方式执行。层执行完毕后，可立即重新分片，以

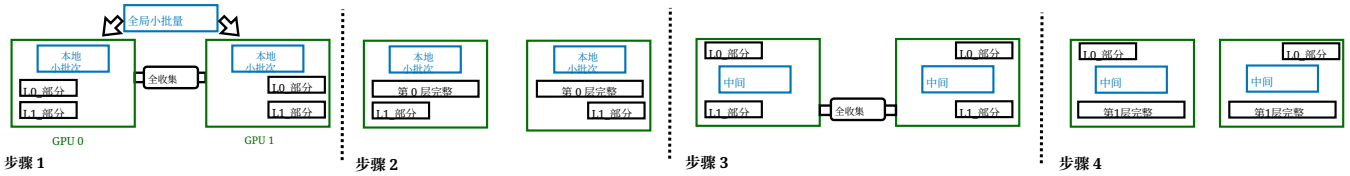


图 9：在一个简单的双层模型上，以全分片数据并行风格执行的片段，该模型被拆分到 2 个图形处理器上。每个阶段都会进行全收集，以便在当前层上执行数据并行，而未激活的层仍分片分布在各设备上。因此，数据并行和模型并行的优势得到了部分融合。

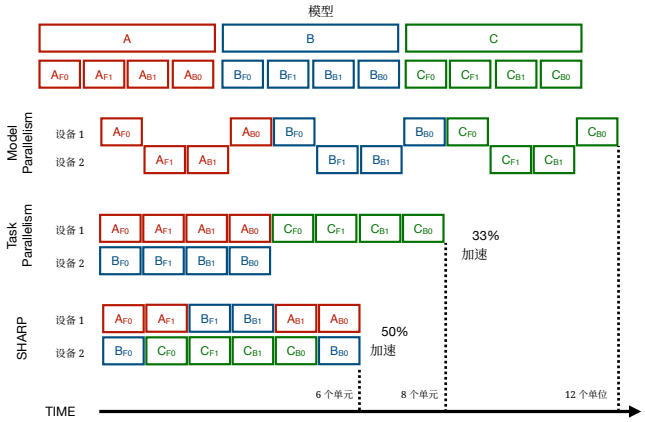


图 9：“分片交替并行”结合了顺序模型并行溢出和任务并行，性能优于标准模型并行和任务并行技术。

重新分配内存占用。反向传播也采用类似的方法。

在全分片数据并行下，每个加速器的内存需求会降低到最低限度：保留单个层所需的内存，以及其余层分片后的内存需求。将单层需求视为常数因子并忽略后，可以将其表示为 $O(n/k)$ 的缩减，其中 n 是原始模型的内存占用，k 是数据并行实例的数量。这使用户能够同时享有数据并行的性能和模型并行的可扩展性。请注意，这会带来大量通信开销——每一层都要执行一次全收集——而且与基于溢出的技术不同，它仍需通过横向扩展来实现可扩展性。

ZeRO-Offload [48] 提出将 FSDP 与每个加速器的溢出相结合，卸载近期不会使用的分片层参数。这显著提升了可扩展性，但也通过 CPU-GPU 通信带来了更多通信开销。ZeRO 致力于将通信与计算重叠起来，但通常不可避免地会造成一定的减速。分析表明，FSDP 的速度比标准数据并行更慢（尽管可扩展性更强，也能运行规模显著更大的模型）。FSDP 的支持者声称，用户可以利用其更强的可扩展性来增大批量大小（从而使执行时间与 DDP 的性能相当），但我们指出，批量大小会影响精度收敛行为。增大批量大小以提升 FSDP 性能可能会导致与我们在讨论异步流水线并行时所述相同的问题（尽管程度较轻）。

3D并行将全分片数据并行与流水线并行和张量并行相结合，既能利用可扩展的数据并行，也能实现沿深度和宽度方向分片的并行执行。这通常表现为在模型的某些部分应用全分片数据并行，在另一部分采用流水线并行，并在更适合按宽度分片的另一段采用张量并行。3D并行通常需要根据模型架构进行大量定制，无法像 Hydra 或全分片数据并行那样开箱即用。尽管如此，Megatron [53] 等系统已成功将其应用于训练许多超大规模模型，例如 Megatron-LM [5] 和 BLOOM [12]。未来，或许可以将 3D 混合并行与 Hydra 的分片任务并行相结合，实现一种新的“4D并行”。

4.4.2 策略搜索。策略搜索系统旨在自动完成在模型中组合并行化技术的过程。近期的一些实例包括 FlexFlow [26] 和 Alpa [60]。

FlexFlow 构建于流水线并行、全分片数据并行和分片任务并行等先进深度学习并行化技术出现之前，因此仅探索了数据并行、张量并行和模型并行，且主要面向卷积神经网络。FlexFlow 构建了设备拓扑图，将加速器建模为节点，将互连（例如 NVLink、PCIe 和 InfiniBand 网络）建模为边。这样，它便能生成混合并行执行策略，并考虑给定设备配置中边之间数据移动的成本。它使用模拟器评估不同的分区策略，通过试运行对算子运行时间建模，并依据边带宽进行理论计算，以估算通信开销。它将模拟器用作预言机，评估算子的不同分区方式。需要注意的是，这种基于“分区”的并行表示方式无法支持利用不同任务独立执行的并行化技术（例如任务并行、流水线并行），但有可能支持全分片数据并行。此外，它没有明确考虑内存可扩展性，也未考虑特定配置下设备内存耗尽的可能性 [11]。

Alpa 更明确地考虑了内存可扩展性，并考量算子间并行（例如模型并行、流水线并行），而不仅仅像 FlexFlow 那样进行算子内分区。它使用整数线性规划公式来确定如何设置并行化策略；如果某个阶段将超出设备内存限制，则会修改执行计划 [60]。通过考虑更广泛的策略搜索空间，这种方法能够实现比 FlexFlow 更好的性能。

这些混合并行化策略非常适合静态、非数据依赖型执行任务（例如非循环神经网络）。然而，它们难以扩展到更动态的任务，例如多模型训练——它们是用于训练的编译器，而非调度器。未来的研究可以考虑弥合这一差距，构建动态混合并行执行器。

4.4.3 推荐模型的模型-数据并行。深度学习推荐模型给实践者带来了独特的挑战，因为它们结合了两种不同的扩展难题。嵌入表非常宽，因此适合按宽度分区，以便进行模型并行执行。顶部深度神经网络计算密集但规模较小，因此最能受益于数据并行。因此，将张量并行应用于模型的表，同时将数据并行应用于深度神经网络的混合策略，能够很好地适用于推荐模型。这种方法已成为完全采用 GPU 加速的深度学习推荐模型训练的标准 [41]，不过，对于 GPU 资源较少的用户而言，我们先前介绍的 CPU-GPU 异构执行方式也很受欢迎。

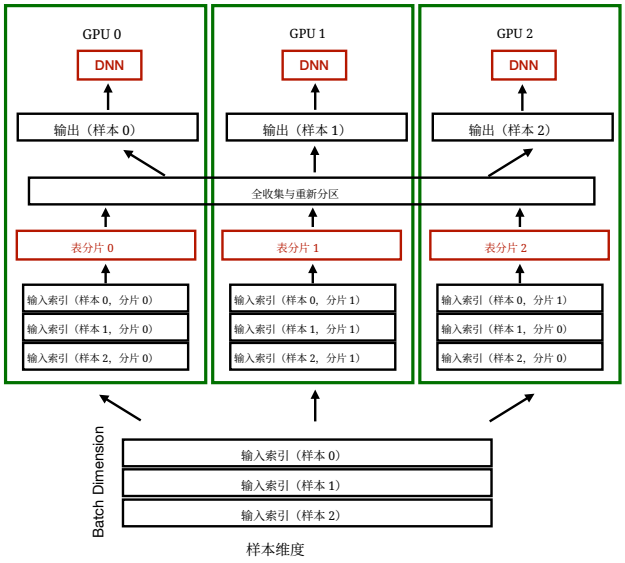


图 10：使用混合模型-数据并行，在 3 个图形处理器上以批量大小为 3 的小批量训练 DLRM 时的数据流和执行模式示意图。

混合并行深度学习推荐模型训练会将嵌入表分片到多个图形处理器上，并在每个图形处理器上放置一份顶部深度神经网络的本地副本。分片后的表会处理沿样本维度分片的输入，随后执行分区全收集，以重新聚合表输出，并将其沿批次维度分配给每个数据并行副本。图 10 展示了这一过程。

这种方法让深度学习从业者能够在神经网络架构中兼顾数据并行和模型并行。通信步骤开销很大，通常会带来沉重的开销 [35]，但并行执行的优势通常足以弥补这一点。

总体而言，混合并行使用户能够在适当情况下结合不同并行策略的优势，从而高效地训练模型。诸如分片任务并行和全分片数据并行从底层兼顾可扩展性与效率，而策略搜索和 DLRM 混合并行则有助于训练模型架构，使其能够满足计算图不同阶段的混合需求。

5 结论与讨论

本文综述了大模型训练系统研究，涵盖从异构执行到混合并行的各种方法。随着从业者不断扩大模型规模，大模型训练系统研究正变得日益重要。在众多领域中，效率和可扩展性已成为深度学习从业者关注的关键问题，而开发这些系统对于推动深度学习进一步发展至关重要。

大模型训练研究仍有多个有待探索的方向。现有的混合并行训练系统所覆盖的搜索空间仍然相对有限，目前用户可用的最复杂配置是 3D 并行，即流水线并行、数据并行和张量并行。未来的混合并行系统可以进一步拓展，引入更多维度，例如用于多模型训练的任务并行和 CPU 溢出，从而有望实现更高的可扩展性和效率。

新的硬件发展也将推动这一领域系统的进步。新型 CPU-GPU 服务器正在开发中，旨在支持 CPU 与图形处理器之间通过 NVLink 进行高带宽通信——与 Tesla GPU 上的 PCIe 3.0x16 互连相比，这将使溢出通信速度提高近 60 倍 [7]。内存容量更大的 TPU 和其他定制硬件，相较于内存受限的图形处理器，也颇具吸引力。随着训练硬件的限制和约束发生变化，大模型训练软件系统的设计及其目标也将随之改变。

当前系统并未针对无服务器执行进行显式优化——无论是在固定的本地集群上，还是在云端提供的机器上，它们都能同样良好地运行。但无服务器执行既带来了新的机遇，也带来了新的挑战。例如，预算约束训练可以让用户指定愿意为某项实验支出的金额，随后由自动并行化系统直接确定应配置哪些资源，以便在成本限制内实现高效训练。面向动态大规模训练的弹性扩展有望带来许多令人振奋的机遇，例如自动扩展混合并行设计的维度，或进行自动评估。

大模型训练系统领域将持续发展，深度学习从业者探索更广泛架构的需求将推动这一进程。随着深度学习对计算和内存的需求不断增长，该领域的重要性也将与日俱增。本文综述的各个研究方向将为未来的新进展奠定基础，以满足该领域不断变化的需求。

参考文献

[1] NVIDIA CuDNN。https://developer.nvidia.com/cudnn。[2] Nvidia V100 Tensor Core GPU。https://www.nvidia.com/en-us/data-center/v100/。[3] 推出原生 PyTorch 自动混合精度，加速 NVIDIA GPU 上的训练。https://pytorch.org/blog/accelerating-training-on-nvidia-gpus-with-pytorch-automatic-mixed-precision/，2020。

[4] 通过增大模型规模加速 Transformer 训练与推理。https://bair.berkeley.edu/blog/2020/03/05/compress/，2020。[5] 使用 Megatron 在 NVIDIA A100 GPU 上进行先进的语言建模。https://developer.nvidia.com/blog/language-modeling-using-megatron-a100-gpu/，2020。[6] 图神经网络系统综述。https://github.com/makemebitter/ gnnsys_servey/blob/master/main_re.pdf，2021。[7] NVIDIA 将在 Hot Chips 上分享 Grace CPU、Hopper GPU、NVLink 交换机及 Jetson Orin 模块的新详情。https://blogs.nvidia.com/blog/2022/08/19/grace- hopper-nvswitch-hot-chips/，2022。[8] M. Abadi、P. Barham、J. Chen、Z. Chen、A. Davis、J. Dean、M. Devin、S. Ghemawat、G. Irving、M. Isard、M. Kudlur、J. Levenberg、R. Monga、S. Moore、D. G. Murray、B. Steiner、P. Tucker、V. Vasudevan、P. Warden、M. Wicke、Y. Yu 和 X. Zheng。Tensorflow：一种用于大规模机器学习的系统。发表于第 *12* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 16*283，2016 265），页码 – 。[9] B. Acun、M. Murphy、X. Wang、J. Nie、C.-J. Wu 和 K. Hazelwood。大规模深度学习推荐模型训练效率研究，2020。[10] M. Adnan、Y. E. Maboud、D. Mahajan 和 P. J. Nair。用于推荐系统训练的异构加速流水线，2022。[11] N. Band。Memflow：具备内存感知能力的分布式深度学习。发表于 *2020* 年 *ACM SIGMOD* 数据管理国际会议论文集，SIGMOD '20，第 2883–2885 页，纽约州纽约市，美国，2020。计算机协会。[12] BigScience。BigScience 语言开放科学开放获取多语言（BLOOM）语言模型，2022。[13] T. B. Brown、B. Mann、N. Ryder、M. Subbiah、J. Kaplan、P. Dhariwal、A. Neelakan- tan、P. Shyam、G. Sastry、A. Askell、S. Agarwal、A. Herbert-Voss、G. Krueger、T. Henighan、R. Child、A. Ramesh、D. M. Ziegler、J. Wu、C. Winter、C. Hesse、M. Chen、E. Sigler、M. Litwin、S. Gray、B. Chess、J. Clark、C. Berner、S. McCandlish、A. Radford、I. Sutskever 和 D. Amodei。语言模型是少样本学习器，2020。[14] A. Castelló、M. F. Dolz、E. S. Quintana-Ortí 和 J. Duato。分布式神经网络模型并行分析。发表于第 *26* 届欧洲 *MPI* 用户组会议论文集，EuroMPI '19，纽约州纽约市，美国，2019。计算机协会。[15] T. Chen、B. Xu、C. Zhang 和 C. Guestrin。以亚线性内存开销训练深度神经网络，2016。[16] J. Devlin、M.-W. Chang、K. Lee 和 K. Toutanova。BERT：用于语言理解的深度双向 Transformer 预训练，2018。[17] S. Fan、Y. Rong、C. Meng、Z. Cao、S. Wang、Z. Zheng、C. Wu、G. Long、J. Yang、L. Xia、L. Diao、X. Liu 和 W. Lin。DAPPLE：一种用于训练大型模型的流水线式数据并行方法，2020。[18] J. Gou、B. Yu、S. J. Maybank 和 D. Tao。知识蒸馏：综述。《国际计算机视觉杂志》，129(6)：1789–1819，2021。[19] A. Griewank 和 A. Walther。算法 799：Revolve：一种用于计算微分反向模式或伴随模式的检查点实现。*ACM* 数学软件汇刊，26(1)：19–45，2000 年 3 月。[20] J. Gusak、D. Cherniuk、A. Shilova、A. Katrutsa、D. Bershatsky、X. Zhao、L. Eyraud- Dubois、O. Shlyazhko、D. Dimitrov、I. Oseledets 和 O. Beaumont。大规模神经网络训练综述，2022。[21] A. Harlap、D. Narayanan、A. Phanishayee、V. Seshadri、N. Devanur、G. Ganger 和 P. Gibbons。PipeDream：快速高效的流水线并行深度神经网络训练，2018。[22] K. He、X. Zhang、S. Ren 和 J. Sun。用于图像识别的深度残差学习，2015。[23] C.-C. Huang、G. Jin 和 J. Li。SwapAdvisor：通过智能交换突破 GPU 内存限制，实现深度学习。发表于第二十五届国际编程语言与操作系统体系结构支持会议（*ASPLOS*）论文集，2020 年 3 月。[24] J. Huang 和 R. A. van de Geijn。Blislab：用于优化通用矩阵乘法的沙盒，2016。[25] Y. Huang、Y. Cheng、A. Bapna、O. Firat、M. X. Chen、D. Chen、H. Lee、J. Ngiam、Q. V. Le、Y. Wu 和 Z. Chen。GPipe：使用流水线并行高效训练巨型神经网络，2018。[26] 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，2018。[27] C. Kim、H. Lee、M. Jeong、W. Baek、B. Yoon、I. Kim、S. Lim 和 S. Kim。torchgpipe：用于训练巨型模型的即时流水线并行，2020。[28] D. P. Kingma 和 J. Ba。Adam：一种随机优化方法，2014。[29] A. Kumar、S. Nakandala、Y. Zhang、S. Li、A. Gemawat 和 K. Nagrecha。Cerebro：一种用于可扩展深度学习的分层数据平台。发表于第 *11* 届创新数据系统研究年会（*CIDR'21*），2021。[30] T. D. Le、H. Imai、Y. Negishi 和 K. Kawachiya。Tflms：通过图重写在 TensorFlow 中支持大型模型，2018。[31] D. Lepikhin、H. Lee、Y. Xu、D. Chen、O. Firat、Y. Huang、M. Krikun、N. Shazeer 和 Z. Chen。Gshard：通过条件计算和自动分片扩展巨型模型，2020。[32] S. Li 和 T. Hoefler。Chimera：利用双向流水线高效训练大规模神经网络。载于《国际高性能计算、网络、存储与分析会议论文集》，第 1–14 页，2021 年。[33] Z. Li、S. Zhuang、S. Guo、D. Zhuo、H. Zhang、D. Song 和 I. Stoica。Terapipe：用于训练大规模语言模型的令牌级流水线并行，2021 年。[34] C. Meng、M. Sun、J. Yang、M. Qiu 和 Y. Gu。通过 TensorFlow 上的 GPU 内存优化训练更深层的模型。载于 NIPS 机器学习系统研讨会论文集，第 7 卷，2017 年。[35] K. Nagrecha。2022 年机器学习平台聚会——Netflix 演讲。https://www.youtube.com/watch?v=qqw2svb4tyo&list=PLs3G-bSut1j_KiIg-z5savMtbGrStrl D&index=3.[36] ¯K. Nagrecha。面向深度学习系统的模型并行模型选择。载于《2021 年国际数据管理会议论文集》。ACM，2021 年 6 月。[37] K. Nagrecha 和 A. Kumar。Hydra：一个用于大型多模型深度学习的系统，2021 年。[38] S. Nakandala、K. Nagrecha、A. Kumar 和 Y. Papakonstantinou。通过增量式和近似计算加速深度卷积神经网络推理。《ACM 数据库系统汇刊》（TODS），45(4)：1–42，2020 年。[39] D. Narayanan、A. Phanishayee、K. Shi、X. Chen 和 M. Zaharia。内存高效的流水线并行深度神经网络训练，2021 年。[40] D. Narayanan、M. Shoeybi、J. Casper、P. LeGresley、M. Patwary、V. Korthikanti、D. Vainbrand、P. Kashinkunti、J. Bernauer、B. Catanzaro 等。使用 Megatron-LM 在 GPU 集群上高效训练大规模语言模型。载于《国际高性能计算、网络、存储与分析会议论文集》，第 1–15 页，2021 年。[41] M. Naumov、D. Mudigere、H.-J. M. Shi、J. Huang、N. Sundaraman、J. Park、X. Wang、U. Gupta、C.-J. Wu、A. G. Azzolini、D. Dzhulgakov、A. Mallevich、I. Cherniavskii、Y. Lu、R. Krishnamoorthi、A. Yu、V. Kondratenko、S. Pereira、X. Chen、W. Chen、V. Rao、B. Jia、L. Xiong 和 M. Smelyanskiy。用于个性化和推荐系统的深度学习推荐模型，2019 年。[42] A. Paszke、S. Gross、F. Massa、A. Lerer、J. Bradbury、G. Chanan、T. Killeen、Z. Lin、N. Gimelshein、L. Antiga、A. Desmaison、A. Köpf、E. Yang、Z. DeVito、M. Raison、A. Tejani、S. Chilamkurthy、B. Steiner、L. Fang、J. Bai 和 S. Chintala。PyTorch：一种命令式风格、高性能的深度学习库，2019 年。[43] P. Patarasuk 和 X. Yuan。面向工作站集群的带宽最优全归约算法。[44] A. Polino、R. Pascanu 和 D. Alistarh。通过蒸馏和量化实现模型压缩。arXiv 预印本 arXiv:1802.05668，2018 年。[45] B. Pudipeddi、M. Mesmakhosroshahi、J. Xi 和 S. Bharadwaj。使用一种新的执行算法以恒定内存训练大型神经网络，2020 年。[46] S. Rajbhandari、J. Rasley、O. Ruwase 和 Y. He。ZeRO：面向万亿参数模型训练的内存优化，2019 年。[47] S. Rajbhandari、O. Ruwase、J. Rasley、S. Smith 和 Y. He。ZeRO-Infinity：突破极大规模深度学习的 GPU 内存墙，2021 年。[48] J. Ren、S. Rajbhandari、R. Y. Aminabadi、O. Ruwase、S. Yang、M. Zhang、D. Li 和 Y. He。ZeRO-Offload：让十亿级模型训练惠及大众，2021 年。[49] M. Rhu、N. Gimelshein、J. Clemons、A. Zulfiqar 和 S. W. Keckler。vDNN：用于可扩展、内存高效神经网络设计的虚拟化深度神经网络，2016 年。[50] A. Sergeev 和 M. Del Balso。Horovod：在 TensorFlow 中实现快速、简单的分布式深度学习，2018 年。[51] N. Shazeer、Y. Cheng、N. Parmar、D. Tran、A. Vaswani、P. Koanantakool、P. Hawkins、H. Lee、M. Hong、C. Young、R. Sepassi 和 B. Hechtman。Mesh-TensorFlow：面向超级计算机的深度学习。载于《神经信息处理系统》，2018 年。[52] J. P. Shen 和 M. H. Lipasti。《现代处理器设计：超标量处理器基础》。Waveland Press，2013 年。[53] M. Shoeybi、M. Patwary、R. Puri、P. LeGresley、J. Casper 和 B. Catanzaro。Megatron-LM：使用模型并行训练数十亿参数语言模型，2019 年。[54] N. S. Sohoni、C. R. Aberger、M. Leszczynski、J. Zhang 和 C. Ré。低内存神经网络训练：技术报告，2019 年。[55] F. G. Van Zee 和 R. A. van de Geijn。BLIS：一种快速实现 BLAS 功能的框架。《ACM 数学软件汇刊》，41(3)：14:1–14:33，2015 年 6 月。[56] L. Wang、J. Ye、Y. Zhao、W. Wu、A. Li、S. L. Song、Z. Xu 和 T. Kraska。Superneurons。载于《第 23 届 ACM SIGPLAN 并行编程原理与实践研讨会论文集》。ACM，2018 年 2 月。[57] P. Wang、K. Nagrecha 和 N. Vasconcelos。用于机器教学的基于梯度的算法。载于《IEEE/CVF 计算机视觉与模式识别会议论文集》，第 1387–1396 页，2021 年。[58] Y. E. Wang、G.-Y. Wei 和 D. Brooks。深度学习用张量处理单元、图形处理器和中央处理器平台基准测试，2019 年。[59] B. Yang、J. Zhang、J. Li、C. Ré、C. R. Aberger 和 C. D. Sa。Pipemare：异步流水线并行深度神经网络训练，2020 年。

[60] L. Zheng、Z. Li、H. Zhang、Y. Zhuang、Z. Chen、Y. Huang、Y. Wang、Y. Xu、D. Zhuo、E. P. Xing、J. E. Gonzalez 和 I. Stoica。Alpa：为分布式深度学习自动化算子间与算子内并行，2022 年。

[61] W. Zhu、C. Zhao、W. Li、H. Roth、Z. Xu 和 D. Xu。Lamp：用于图像分割、具有自动模型并行功能的大型深度神经网络，2020。