Angel-PTM：腾讯可扩展且经济高效的大规模预训练系统

Xiaonan Nie
北京大学
xiaonan.nie@pku.edu.cn

Yi Liu
腾讯公司
callbackliu@tencent.com

Fangcheng Fu
北京大学
ccchengff@pku.edu.cn

Jinbao Xue
腾讯公司
jinbaoxue@tencent.com

Dian Jiao
腾讯公司
focusjiao@tencent.com

Xupeng Miao
卡内基梅隆大学
xupeng@cmu.edu

Yangyu Tao
腾讯公司
brucetao@tencent.com

Bin Cui
北京大学
bin.cui@pku.edu.cn

arXiv:2303.02868v1 [cs.LG] 6 Mar 2023

摘要

近年来，大规模预训练模型，尤其是 Transformer 模型，取得了前所未有的成就。腾讯公司旗下的许多产品和服务，例如微信、QQ 和腾讯广告，都已采用预训练模型以发挥其优势。在本文中，我们介绍了 Angel-PTM，这是一种专为 Transformer 模型预训练和微调而设计的高效深度学习系统。Angel-PTM 能够借助分层内存高效地训练超大规模模型。Angel-PTM 的核心设计包括通过页 抽象实现细粒度内存管理，以及一种统一调度方法，用于协调计算、数据移动和通信。此外，Angel-PTM 支持利用 SSD 存储实现极大规模的模型缩放，并实现了无锁更新机制，以应对 SSD I/O 带宽瓶颈。实验结果表明，在最大模型规模方面，Angel-PTM 比现有系统最高提升 114.8%；在训练吞吐量方面，最高提升 88.9%。此外，在数百个 GPU 上对 GPT3-175B 和 T5-MoE-1.2T 模型进行的实验验证了 Angel-PTM 的强大可扩展性。

1 引言

近年来，Transformer模型等大规模预训练模型在计算机视觉 [11, 29, 47],自然语言处理 [7, 9, 41, 42],语音识别 [10, 54] 和生成式人工智能 [1, 48] 等多个领域取得了显著进展，超越了传统机器学习模型，成为最先进的方法。例如，Chat-GPT [1] 能够生成类人文本，并以令人印象深刻的准确性和效率执行各种自然语言处理任务。Transformer模型的成功归功于其能够自动学习并提取数据的层次化表示，因此非常适合处理复杂任务 [6]。

为了实现更好的性能，人们不断努力扩大模型规模——旗舰级自然语言处理模型的规模每 2 年就会增长 240× [17], , Kaplan 等人 [25] 指出，为了提升模型质量，模型规模持续增长的趋势还将延续。这种模型规模的爆炸式增长不可避免地会增加计算和内存成本，并且训练大型 Transformer模型的成本也因此变得极其高昂。例如，微软训练 Megatron-Turing NLG 530B 时，全部使用了 4480 个 A100 80G GPU [49], , 仅购买这些计算资源就几乎要花费 7000 万美元。

毋庸置疑，为了在实际应用中充分发挥Transformer模型的卓越能力，同时适应模型规模的快速演进，企业有必要在这个时代重新思考并重新设计高效的深度学习系统。结合腾讯公司的应用场景和需求，我们希望指出深度学习系统的两个关键特征。

• **易于使用且易于扩展**。在实际生产应用中，大多数用户是深度学习研究人员或数据科学家，擅长设计面向特定任务的模型架构。相比之下，他们通常缺乏以分布式方式部署并加速模型训练过程所需的专家知识或经验。因此，深度学习系统应只需进行少量修改，即可将训练任务并行化。此外，由于生产集群本质上是多租户的，可用硬件资源会不断变化。因此，我们追求无缝的可扩展性。换言之，当用户希望调整其任务所使用的资源量时，无需重新配置并行方案。

• **高效且经济**。由于模型和数据集的规模惊人，训练大规模模型极其耗时。因此，如何实现更高的训练效率，对于预训练模型的实际部署至关重要。除了运行时间之外，我们还应尽最大努力提高硬件利用率。毫无疑问，如果能够使用尽可能少的资源高效训练大规模模型，就能使预训练模型更加经济。

我们观察到，现有的深度学习系统（如 Megatron- LM [35] 和 DeepSpeed [43, 46], ）无法实现这些特性。首先，许多系统采用复杂的并行策略（如张量并行和流水线并行）来扩展训练任务规模，这要求用户深入了解如何将模型拆分到可用的加速器（如 GPU设备）上，以优化分布式计算效率。这会增加部署和扩展训练任务所需的额外工作。其次，我们发现，现有系统采用粗粒度的内存管理方式

在训练期间分配和释放内存，因此在训练大规模 Transformer模型时，不可避免地会受到内存碎片和资源利用率低的影响，正如我们将在第 3 节中进一步深入分析的那样。

在这项工作中，我们开发了 Angel-PTM，这是一种全新的深度学习系统，旨在支持腾讯日益兴起的大规模预训练模型应用。Angel-PTM 的主要贡献总结如下：

• 我们分析了腾讯大规模模型训练任务的特征与需求，并提出了 Angel-PTM 的底层设计以满足这些需求。该设计整合了数据并行、参数分片和分层内存，从而提升易用性，并使系统能够透明地扩展至不同数量的 GPU。

• 为减少内存碎片并充分利用内存和带宽，我们提出了细粒度页抽象，并在页级管理模型状态，包括内存分配、释放、移动和通信。此外，我们设计了统一调度器，并结合基于细粒度生命周期的调度方法，以整体方式动态管理这些操作，从而实现高效训练。

• 为了支持将模型扩展到极大规模，我们集成了 SSD 存储，并设计了无锁更新机制，以消除 SSD I/O带宽瓶颈。

• 我们对多种具有代表性的大规模 Transformer模型进行了评估。结果表明，Angel-PTM 实现了
与现有系统相比，最大支持模型规模提升了.114.8%，吞吐性能提升了 88.9%。实验还验证了 Angel-PTM 在数百个 GPU 上进行训练时具备近线性可扩展性。

Angel-PTM 已在腾讯部署约一年，为众多产品和服务提供支持，包括微信、QQ、腾讯游戏、腾讯广告和腾讯云，其大模型涵盖自然语言处理、计算机视觉和跨模态任务等多个人工智能领域。此外，Angel-PTM 还支持了混元系列大模型的训练，其中混元-1T模型在 CLUE基准测试的总排名中位居第一[1]。

2 背景

2.1 深度学习中的内存管理

随着模型规模的不断增大，GPU 内存管理
如今已成为影响深度学习模型性能和可扩展性的关键因素 [7, 25, 36]。本节将简要介绍训练期间的内存消耗情况，以及当前深度学习框架如何对其进行管理。

深度学习训练。深度学习模型由多个数学函数层组成，每一层都将前一层的输出作为输入并生成输出。因此，训练的计算过程可以表示为计算图，其中每个节点代表一个操作（例如矩阵乘法），每条边表示一个张量或依赖关系 [3]。DNN 的计算图通常首先使用拓扑排序算法转换为操作列表，然后

[1]https://cluebenchmarks.com/rank.html

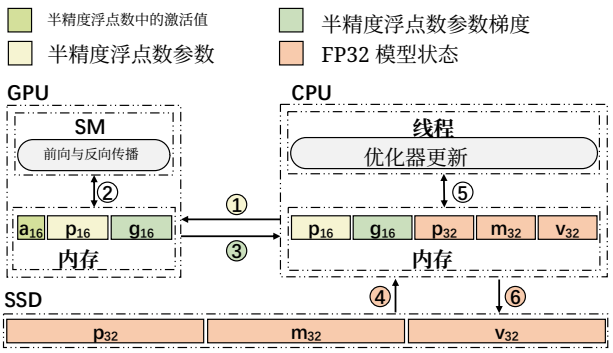

图 1：分层内存上的训练工作流程示意图。

执行器按顺序执行这些操作。为了获得令人满意的模型质量，训练过程需要进行大量的前向传播和反向传播。在前向传播过程中，一批训练数据会经过计算图，并利用模型参数为每个操作生成激活值。随后，使用损失函数将最终输出与预期值进行比较。在反向传播过程中，误差值会沿计算图反向传播，以分别计算激活值和参数的梯度，其中参数梯度还会由优化器用于更新模型参数。总之，训练期间的内存主要由参数及其梯度、激活值及其梯度，以及优化器状态占用。其中，参数和优化器状态会在训练期间保留，而激活值、激活值的梯度以及参数的梯度会在每次迭代中动态生成并释放。在本文余下部分，为简便起见，我们将参数和优化器状态统称为“模型状态”。

混合精度训练。为了在不牺牲模型质量的情况下减少计算和内存需求，Micikevi- cius 等人 [34] 提出了用于训练的混合精度 技术。如图 1 所示，在计算之前，参数会被转换为半精度格式（即 FP16 或 BF16 变体），因此激活值和两种梯度都会以 FP16 方式进行计算。同时，模型状态以单精度格式（即 FP32）存储，以保持模型质量。随着模型规模的快速增长，混合精度 已成为大规模模型训练和部署的事实标准。

现有框架中的内存管理。现有的深度学习框架（例如 PyTorch [40]、TensorFlow [3] 和 Hetu [31]）采用通用的内存管理方法，将 GPU 内存置于独立的内存池中进行管理，以响应执行器的请求，例如内存分配和内存释放。例如，TensorFlow 使用最佳适配内存分配（BFC）算法管理 GPU 内存，从而减少频繁内存分配和释放造成的内存碎片。该算法仅分配必要的内存量，尽力减少 GPU 内存的浪费。与其他内存分配算法相比，BFC 算法可能需要更长时间才能找到可用模块，但它非常适合 GPU 内存有限的系统。

表 1：采用 Adam 优化器进行混合精度训练时，单个 Transformer 层的内存占用。

模块	层	参数（B）	激活值（B）	优化器状态。(B)
Attn	线性层(Q,K,V)	$12d_m^2$	$12bsd_m$	$36d_m^2$
	矩阵乘法	-	$4bs$	-
	缩放-掩码 Softmax	-	$4bs$	-
	矩阵乘法	-	$4bsd_m$	-
	线性层	$4d_m^2$	$4bsd_m$	$12d_m^2$
	Add	-	$4bsd_m$	-
	层归一化	$4d_m$	$4bsd_m$	$12d_m$
FFN	线性层	$4d_md_{ffn}$	$4bsd_{ffn}$	$12d_md_{ffn}$
	GeLU	-	$4bsd_{ffn}$	-
	线性层	$4d_md_{ffn}$	$4bsd_m$	$12d_md_{ffn}$
	Add	-	$4bsd_m$	-
	层归一化	$4d_m$	$4bsd_m$	$12d_m$
总计		$16d_m^2$ $+8d_md_{ffn}$	$40bsd_m$ $+8bsd_{ffn}$	$48d_m^2$ $+24d_md_{ffn}$

用于训练的分层内存。为了满足大型模型训练的内存需求，许多框架尝试在 GPU 服务器中引入分层内存。具体而言，为了解决基于 Transformer 的预训练模型的内存消耗问题，研究人员提出将优化器状态和计算转移至 CPU 内存和 SSD 存储 [12, 45, 46]。我们在图 1 中展示了使用分层内存进行训练的工作流程。GPU（1）从 CPU 获取参数，（2）在 GPU 上执行前向和反向计算，然后（3）将计算得到的梯度发送回 CPU。CPU（4）从 SSD 存储加载优化器状态，（5）在 CPU 上执行优化器更新，并且（6）将优化器状态存储在 SSD 存储中。

2.2 Transformer 的内存占用

一个 Transformer层由一个自注意力网络和一个逐位置前馈网络（FFN）堆叠而成，并在这两个子层上分别采用残差连接，随后接一个归一化层 [5]。下面，我们将在广泛使用的混合精度训练和 Adam 优化器场景下，近似计算 Transformer层中各组件的内存占用，其中输入数据为 $X \in \mathbb{R}^{b\times s\times d_m}$。具体而言，$b$ 是批量大小，s 是序列长度，d_m 是嵌入维度，d_{ffn} 是前馈神经网络隐藏维度。内存占用结果汇总于表 1 中。为简化计算总大小，我们忽略较小的张量，例如层归一化的参数。

自注意力。注意力模块 [50] 能够捕捉序列中词元之间的依赖关系，在序列建模中表现出色。如公式 1 所示，它首先通过三个线性函数分别将输入 X 线性投影为查询（Q）、键（K）和值（V），其中 $\{W_Q, W_K, W_V\} \in \mathbb{R}^{d_m\times d_m}$，该层中参数的总占用量为 $3 \times$ (Q, K, V)（前向和反向）（FP16，$^{\times 2}$2 字节）$\times d_m \times d_m = 12d_m^2$。同样，激活值为 $\{Q, K, V\} \in \mathbb{R}^{b\times s\times d_m}$，其占用量为 3(Q, K, V) $\times 2$（前向和反向）$\times 2$（FP16，2 字节）$\times$ $b \times s \times d_m = 12bsd_m$。模型状态（优化器状态）的占用量为 3(Q, K, V) $\times 3$（主参数、动量、方差）$\times 4$（FP32，4 字节）$\times d_m \times d_m = 12d_m^2$。相同的计算方法也适用于其他层；由于篇幅有限，我们将在表 1 中直接给出结果。经过这一线性层后，模型将计算每个查询-键（Q-K）对之间的注意力分数（形状：$b \times s$）；这些分数通过点积（矩阵乘法）操作以及缩放掩码 Softmax操作获得。为提高时间效率并节省内存，缩放、掩码（可选）和 Softmax 这三种操作通常会通过内核融合技术合并为缩放掩码Softmax操作 [51]。注意力向量（形状：$b \times s \times d_m$）通过对注意力分数和值进行加权求和得到 V。最后，自注意力层对注意力向量应用线性变换（$W \in \mathbb{R}^{d_m\times d_m}$），生成输出（形状：$b \times s \times d_m$）。

$$\texttt{Attn}(X) = \texttt{Softmax}\left(\texttt{Mask}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_k}}\right)\right)(XW_V) \quad (1)$$

加法和层归一化。在自注意力层的输入（形状：$b \times s \times d_m$）与输出（形状：$b \times s \times d_m$）之间采用残差连接。随后，对加法操作的输出（形状：$b \times s \times d_m$）应用归一化层。FFN模块也执行相同的变换，我们在公式 2 中对其进行了形式化表示。此外，LayerNorm层的参数及其梯度大小为 $4d_m$，分别对应权重和偏置；与其他部分相比，这部分可以忽略不计。

$$y = \texttt{LayerNorm}(f(x) + x), \quad where\ f \in \{\texttt{Attention}, \texttt{FFN}\} \quad (2)$$

前馈网络。如公式 3 所示，前馈网络（FFN）层通过两个全连接（FC）层对输入进行线性变换，两层之间由 GeLU 激活函数隔开 [19]。具体而言，第一个 FC 层（$W_1 \in \mathbb{R}^{d_m\times d_{ffn}}$）将输入投影到一个维度更高的新空间，使模型能够捕捉单个词元内部更复杂的关系；第二个 FC 层（$W_2 \in \mathbb{R}^{d_{ffn}\times d_m}$）则将维度缩减回原始大小，从而确保模型输出稳定，并避免整体架构过于复杂。

$$\texttt{FFN}(x_s) = W_2 \cdot \texttt{GeLU}(W_1 \cdot x_s) \quad (3)$$

内存使用分析。根据表 1，我们可以估算任意仅解码器 Transformer模型的内存利用率，其中不考虑嵌入_查_表和损失函数。对于 GPT-3 175B [7], , 当批量大小（b）为 1、序列长度（s）为 2048、嵌入隐藏层大小（d_m）为 12288、前馈神经网络隐藏维度（d_{ffn}）为 49152 时，参数、激活值 和优化器状态 分别占用 648GB、162GB 和 1944GB。为满足大型模型训练的内存需求，需要使用多GPU，并采用并行策略进行分布式训练。

2.3 分布式训练

通过在多GPU之间划分模型参数及其计算过程，可以显著加速训练，使研究人员能够在更短的时间内训练大规模Transformer模型

。本节将全面概述目前已广泛应用于Transformer模型的并行策略。

数据并行与零冗余优化。 在数据并行（DP）中，训练样本被划分到多个设备上，而模型参数和优化器状态则在这些设备上重复存储 [28]。每个设备使用本地小批量数据执行前向和反向传播，得到参数梯度；随后通过所有设备间的同步（例如全归约）对梯度求平均。最后，每个设备根据同步后的梯度独立更新模型参数和优化器状态。然而，传统数据并行要求每个设备都维护一份完整的模型状态副本，对于大规模Transformer模型而言，内存效率低下。为了减少内存占用，Rajbhandari 等人 [44] 提出了零冗余优化（ZeRO）技术，将模型状态均匀划分到所有设备上。具体来说，使用 N 个设备进行训练时，每个设备仅存储并更新 $1/N$ 的模型状态。然而，在每次训练迭代中，都需要额外进行一轮全收集通信，以确保每个设备都能获取完整的更新后参数，从而完成传播。总而言之，采用 ZeRO 的数据并行提高了内存效率，但代价是增加了通信开销。

模型并行与混合并行。模型并行将模型拆分到多个设备上，并以分布式方式执行前向传播和反向传播。Megatron-LM [35] 提出了张量并行（TP），将注意力网络中的查询、键和值矩阵按行或按列进行划分，从而利用多头注意力固有的并行性。在流水线并行（PP）中，模型被划分为一系列阶段，每个阶段在单独的设备上执行。Huang 等人 [20] 提出了批次拆分流水线算法，并在多个GPU上实现了近乎线性的加速。混合并行是指结合两种或更多并行策略以提高训练效率，这需要同时考虑计算、通信和存储这三个方面。Miao 等人 [32] 构建了一种基于决策树的优化方法，可自动为模型的每一层找到最优的混合并行策略。

3 动机与系统设计

3.1 腾讯的使用场景

通过从腾讯最大的机器学习平台（即 Angel [23,24, 57]）收集大量训练任务数据，我们识别出两类常见的使用场景，分别是预训练和微调。这些任务大多涉及 Transformer模型的大规模训练，因为 Transformer 架构彻底改变了自然语言处理，并能够高效处理具有长距离依赖的序列数据。每类任务都有其独特的特征和主要目标，我们将在下一节对这些类别进行详细分析。此外，我们还将简要介绍相应的系统优化，以提升其效率和效果。

预训练。 预训练是指在大量未标注数据上训练大型 Transformer 模型的过程，以学习丰富多样的特征，进而应用于各种下游任务，例如问答和机器翻译。考虑到模型规模和数据集的大小，预训练任务极其耗时且占用大量内存。同时，Kaplan 等人 [25] 指出，预训练模型的模型质量会随数据量、模型规模和计算量的增加而按照幂律提升，这进一步增加了预训练任务的需求。

分析平台的日志信息后，我们发现，尽管预训练任务需要使用数百甚至数千个 GPU 进行数周训练，但它们仅占任务总数的约 10%。这是因为腾讯的研究人员倾向于联合训练一个大规模 Transformer模型，作为共享基础模型。我们还观察到，预训练具有两个主要特征：

- **可扩展性低效**。 在预训练任务运行期间，用户可能会暂停任务并申请更多 GPU，以便更快地获得实验结果。然而，在许多情况下，增加 GPU 后，GPU利用率和训练吞吐量反而会下降，具体取决于模型的分布式训练策略。以一个实际案例为例：使用 Megatron-LM 的混合并行策略，在 72 个 GPU 上训练一个 64 层的 GPT 模型，比在 64 个 GPU 上训练更慢。正如第 1 节所述，腾讯的许多研究人员并不熟悉分布式计算，因此无法相应地调整并行策略。
- **故障与恢复**。 随着更多 GPU 加入，平均故障间隔时间（MTTF）也会相应缩短。鉴于 GPU 数量庞大且训练时间漫长，预训练任务很可能遭遇 GPU 故障，并且故障后需要重新启动。

微调。 微调是指对预训练模型进行调整，使其适应使用领域专用数据的特定下游任务，例如 Instruct-GPT [39] 中的监督微调（SFT）阶段。针对特定下游任务微调预训练模型后，研究人员希望将所得模型部署到实际产品中。因此，他们需要仔细调整超参数并反复进行实验，以获得尽可能好的性能。此阶段需要反复试验，可能需要多次迭代才能获得满意的模型。

分析平台的日志信息后，我们注意到，微调任务约占任务总数的 90%。每个微调任务同样需要大量 GPU，但其运行时间比预训练任务更短（通常以小时计）。此外，我们发现微调主要具有以下三个特征：

- **GPU利用率低效**。 由于微调任务中的下游数据集规模较小，并且为防止过拟合，通常会采用较小的批量大小。然而，这会导致大量时间耗费在分布式通信上，从而降低昂贵 GPU 计算单元的利用率。
- **响应时间长**。任务队列中有大量微调任务，每项任务都需要一定数量的 GPU 资源。由于集群资源有限，大多数任务都必须等待，等待时间长达数小时。考虑到大多数微调任务通常只需几个小时，这种

表 2：GPT3 单层内张量大小的分布。

张量大小（MB）	3072	2304	1152	768	576
数量	4	6	4	20	12
张量大小（MB）	288	0.375	0.046875	0.0234375	
数量	8	4	6	4	

响应时间长的情况严重阻碍了高效应用的开发。

3.2 系统设计

基于上述观察和分析，我们采用以下三种策略作为系统的底层设计，以应对这些挑战。

数据并行。 在预训练过程中，训练任务所需的 GPU 数量是动态变化的，因此我们的分布式策略必须易于扩展，并具备良好的可扩展性。尽管在某些情况下，模型并行的吞吐量表现可能更好，但并行策略的设计需要专家知识，而且不同并行度之间难以迁移。因此，我们决定选择数据并行作为基础分布式方案。

参数分片。 在数据并行中，每个 GPU 都需要存储完整的模型状态，这使得支持大模型训练变得困难。为了解决这一问题，我们采用 ZeRO [44], 提出的参数分片方法，将每个参数均匀地拆分到多个 GPU 上。需要计算某个参数时，通过全收集操作获取完整参数。该方法显著降低了每个 GPU 的内存需求，使我们能够训练规模更大的模型。

分层内存。 响应时间长和资源利用率低的问题，主要是由于微调任务数量庞大，以及相较于较小的批量大小，这些任务使用的 GPU 数量过多。为解决这些问题，我们在 GPU 服务器中引入异构存储，从而减少微调任务所需的 GPU 数量。此外，利用异构存储还可以让预训练任务在使用相同数量 GPU 的情况下训练更大的模型。

确定系统的底层设计后，我们发现，仅仅结合这些策略仍会导致现有系统的资源利用率不足 [43]。因此，我们首先根据表 1 中给出的公式，分析 GPT3-175B 模型在训练期间单个 Transformer 层中张量大小的分布。结果总结于表 2，其中

这表明张量大小差异很大，从 3072MB 到 0.02MB 不等。我们发现，这主要会带来两种低效问题。

内存利用率不足。 训练时，模型状态需要在不同的存储空间之间移动。然而，由于张量大小不一致，无法直接重复利用存储空间。相反，分配器需要重新申请并释放内存，从而产生内存碎片。随着训练过程的持续进行以及模型状态的不断移动，内存碎片会越来越多，导致内存利用率低下。

带宽利用率不足。模型状态需要通过 PCIe 在 CPU 和 GPU 之间移动，或通过网络通信在 GPU 之间移动。对于大型张量，例如

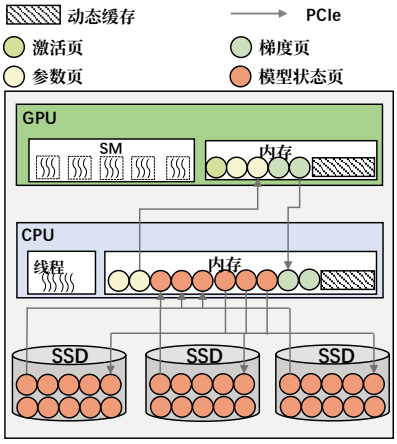

图 2：Angel-PTM 的系统架构。

表 2 中的 3072MB 张量，GPU 中必须有足够的空间才能启动通信。在此之前，通信带宽处于闲置状态，导致带宽利用率低下。

为了解决这些低效问题，我们首先在第 4.1 节提出基于页的内存组织，以提高内存利用率；然后在第 4.2 节设计统一调度器，以充分利用带宽。此外，我们还设计了无锁更新机制，以解决模型扩展到极大规模时 GPU 利用率不足的问题。

4 ANGEL-PTM

我们的系统 Angel-PTM 面向腾讯的研究人员和开发者，旨在帮助他们设计并试验大规模 Transformer 模型。图 2 展示了系统架构：我们在页 级别实现细粒度内存管理，并采用动态缓存技术以充分利用内存。在以下各小节中，我们将分析现有系统的局限性，并介绍解决这些问题的方法。

4.1 基于页的内存组织

内存利用率不足。 现有系统由于内存管理粒度较粗，存在内存碎片问题。例如，DeepSpeed 使用 PyTorch 原有的内存管理机制进行卸载和重计算，这会频繁分配和释放张量。正如第 3 节所述，由于这些张量的大小不一，因此会产生空间碎片。Patrick-Star [12] 以数据块而非张量为单位管理 GPU 内存，其中数据块的大小必须大于模型训练所用的最大张量。这样也会导致每个数据块内部出现内存碎片，并降低通信与计算重叠的效率。

为减少内存空间组织的碎片并提高内存移动效率，我们提出了页 抽象，它作为异构存储中内存操作的最小单位，涵盖内存分配、释放、移动和远程通信。模型状态中的每个张量由若干页组成。

```
struct 页{
  /* 页       信息 */
  void*   data_ptr;
  大小_吨 总计_字节;
  大小_吨   可用_字节;
  // 设备_映射: {0:      GPU, 1: CPU, 2: SSD}
  大小_吨 设备_索引;
  // 标识符  张量的          在此      page
  大小_t 张量_标识符[2];
  // 已占用字节数       每个张量的
  大小_t   张量_字节[2];

  /* 页面     接口          */
  // 分配          所需字节数          用于第 id 个张量
  void 分配（大小_t          所需_字节,          大小_t id）;
  // 释放 id-th 张量的空间
  void释放（大小_tid）;
  // 移动此项       异步将页发送到目标设备
  voidmove(size_t target_设备_index);
  // 异步将此页发送到第 id 个服务器
  void 发送(大小_t id);
  // 从第 id 个服务器接收内容
  void receive(size_t id);
};
```

图3：页抽象。

页 **抽象**。如图3所示，页 抽象包含若干关键信息，包括指向实际内存数据的指针、页中的总字节数、可用于下一次内存分配的字节数，以及页当前所在设备的索引。此外，每个页可以关联一个或多个张量，并记录其唯一标识符及各张量占用的内存量。为简化内存管理，我们决定将每页可包含的张量数量限制为最多两个。我们将在下文结合对最佳页面大小的详细讨论，分析这一决策背后的原因。

此外，页还提供了多个接口，用于访问和操作其中存储的数据。这些接口使开发者能够对页对象执行多种操作，包括为特定张量分配和释放内存、在异构内存之间移动页，以及在不同服务器之间发送和接收页。

最佳页**大小**。页结构的最佳大小是一个关键因素，它关系到如何平衡内存管理效率与整体吞吐量之间相互竞争的需求。一方面，如果页过大，页中将共存大量张量，增加管理的复杂性并造成空间浪费。另一方面，如果页过小，由于带宽利用不足，数据移动的开销会随之增加。因此，我们认为，能够充分利用 PCIe 带宽的最小页大小最适合我们的系统，即 4MB。

事实上，根据我们在训练过程中的观察，绝大多数模型状态都大于 4MB，这一点也在表 1 中得到了验证。因此，通过仔细排列这些张量，我们可以确保每页最多关联两个张量，从而大幅降低管理复杂度。对于小于 4MB 的张量，为了简化处理，我们允许每个张量独占一个

```
结构体张量{
  /* 张量信息 */
  大小_t    id;
  向量 <页>        页_列表;
  大小_t    数据类型;
  大小_t*   形状;
  大小_t    设备_索引;

  /* 张量       接口          */
  void 分配（大小_t* 形状,          大小_t 数据类型);
  void 释放();
  void 移动(大小_t 目标_设备_索引);
  void 合并();
};
```

图4：我们系统中的张量结构。

页，因为它们仅占整体内存利用率中极小的一部分。

张量**管理**。张量结构是我们系统中的一种基础数据结构，用于表示由至少一个页组成的数值数据多维数组。如图4所示，它包含若干关键信息，例如与张量关联的唯一 ID、数据类型、形状以及当前设备索引。[2] 张量结构还提供了一组用于内存管理的接口，例如分配具有特定形状的张量或释放其数据。此外，它还提供了一个用于在异构内存之间移动数据的显式接口。由于不同页的空间可能不连续，张量结构提供了合并接口，使它们保持连续。

4.2 统一调度器

带宽利用率不足。由于模型状态既要通过 PCIe 通信在服务器内的异构内存之间传输，也要通过网络通信在不同服务器之间传输，因此会产生显著的时间开销，并给设计有效的通信方案带来挑战。如何充分利用 PCIe 和通信带宽，对于实现高吞吐量至关重要。为此，必须尽量减少内存交换量，并最大限度地提高通信与计算的重叠程度。DeepSpeed 等现有系统采用静态数据分区和数据传输调度，这可能会降低整体训练速度。这是因为即使 GPU 内存充足，这些系统仍会将全部优化器状态和更新操作传输到 CPU，造成不必要的数据移动。

为了高效利用 GPU 服务器内部层次复杂的网络资源，我们在 页 抽象之上设计了一种统一调度方法，以实现高效训练。如图5所示，用户定义模型后，追踪器会获取张量访问模式以及每个张量的生命周期（详见第 5 节）。随后，统一调度器将这些统计信息作为输入，并在训练过程中于恰当的时间调度每个操作。其核心思想在于，深度学习模型的训练本质上是迭代的，因此我们可以根据张量访问模式对算子进行调度。具体而言，统一调度器通过分析每个张量的生命周期及其大小，

[2]当张量尚未准备好进行计算时（即其部分页需要从异构内存或其他服务器获取），我们将设备索引设为 -1。

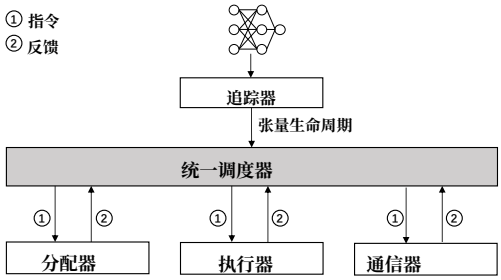

图5：Angel-PTM 中 UnifiedScheduler 及相关组件示意图。

推断出启动每个操作的最佳时机，包括调用分配器移动张量、调用执行器执行 GPU 计算，以及调用通信器进行 GPU间通信。

张量生命周期。 张量的生命周期是指在一次训练迭代中，从首次访问到最后一次访问所持续的时间。深度学习训练的关键特征是其迭代性质，这意味着计算操作会依次、迭代地插入计算流中。这使我们能够针对计算优化数据移动和通信的顺序，并确保每项计算所需的数据都能在恰当的时间就绪，从而减少空闲时间并提高内存效率。此外，由于计算是在张量级别执行的，张量的分配和释放也可以在此级别完成。使用逻辑ID而非实时信息跟踪生命周期，可以简化调度过程。利用张量生命周期信息，我们可以整体优化计算、数据移动和通信的调度。

统一调度器。 统一调度器是 Angel-PTM 中的关键组件，负责协调分配器、执行器和通信器等其他组件的活动。具体而言，它接收来自追踪器的生命周期信息和访问模式，并动态管理内存、计算和通信，通过最大限度地重叠不同资源并减少不必要的操作，实现高训练吞吐量。

首先，我们必须确定一种有效的方法，将模型状态分配到不同的内存设备（如 GPU 内存、CPU 内存和 SSD 存储），并将计算任务分配到不同的计算设备（如 CPU 和 GPU）。这是一个复杂的 NP 规划问题，对于大规模模型无法直接求解。为了解决这个问题，我们采用启发式方法，并结合经验信息来辅助设计策略。具体而言，Transformer 模型的前向和反向计算主要由 FP16 矩阵乘法构成，这类计算计算密集度高、内存需求较低。相比之下，优化器更新计算由 FP32 矩阵加法构成，内存密集度高、计算耗时较短。鉴于 GPU 具有强大的计算能力但内存有限，我们优先将前向和反向计算放在 GPU 上，将优化器更新计算放在 CPU 上。SSD 存储也将

算法 1： 基于细粒度生命周期的调度

输入： $model$：层列表，其中 l_i 是 $i\text{-}th$ 层
$traces$：每个张量的追踪记录列表
$gpuMemory$：GPU 内存总量
输出： S：任务列表，每个任务包含 {operation, page, trigger_id }

/* 阶段 1：优先处理将任务_移至_GPU */
$wait_stack$ ={ };
用于 $l_i \in M$ **执行**
  for $page \in l_i.param.page_list$ **do**
    S.append({move_to_gpu, $page$, 0});
为 $l_i \in M$ **执行**
  当 获取_可用_内存$(S, traces)$ < 大小(l_i) **时**
    $task \leftarrow$ 从 S中弹出最后一个移动任务;
    $wait_stack$.push_回($task.page$);
  对于 $page \in l_i.param.page_list$ **执行**
    S.append({all_聚合, $page$, i});
  S.append({计算, l_i, i});
  当 $wait_stack$.non_empty()且
   获取_可用_内存$(S, traces)$ > 页_大小 **执行**
    $page \leftarrow wait_stack$.弹出_回;
    S.追加({move_到_GPU, $page$, i});

/* 阶段 2：推进所有_集合任务，使其与之前的计算重叠（如果没有发生内存不足（OOM）） */
对于 $task \in \{t | t \in S, t.operation ==$ 所有_聚合$\}$ **执行**
  $task \leftarrow$ 从 S 中 $task$ 弹出;
  $id \leftarrow$ 获取能够在不发生显存溢出的情况下触发的最早 ID $task$;
  $S \leftarrow$ 将 {all_聚合, $task.page, id$} 插入 S;
返回 S;

用于存储 FP32 优化器状态，以扩大模型规模。采用这种方法时，需要将 FP16 参数从 CPU 传输到 GPU，随后再将 FP16 梯度从 CPU 传回 GPU。必须仔细安排这些数据传输，使其与计算重叠，避免产生额外开销。同时，我们利用重计算技术 [8] 进一步缓解 GPU 内存压力：在前向传播中释放部分激活值，然后在反向传播中重新执行相应的前向计算以重新生成这些激活值。

其次，我们利用缓存技术，充分发挥 GPU 在先前分配后的高速内存和计算能力。例如，如果有足够的空间，我们会保留一部分 GPU 内存作为缓存，用于存储一段 CPU 优化器状态。此外，我们将相关的 CPU 计算移至 GPU，从而减少内存传输并加速计算，整体提升训练速度。此外，我们根据每个模型的张量生命周期信息动态确定缓存大小，确保训练过程中不会发生 GPU 显存溢出错误。对于中等规模模型，我们可以在 GPU 上存储并计算大量张量。这样的缓存技术使我们能够最大限度地提高 GPU 内存利用率，并尽量减少数据传输开销。

如算法1所示，调度方法以输入模型、每个张量的跟踪信息以及总

GPU 内存为输入，生成一个任务调度方案，以协调系统的其他组件。该算法包含两个阶段。在阶段 1 中，我们优先将数据移至 GPU：在调度开始处，为每个张量的每一页插入 move_to_gpu 任务（第 3–5 行）。这是基于我们已有的认识：CPU-GPU 数据传输速度（32GB/s）低于 GPU-GPU 通信速度（200GB/s）。随后，我们遍历每一层，按需插入所有_gather 任务和计算任务（第 10–12 行）。如果当前可用内存不足以运行当前层，我们将移除最后一个移动任务，直至满足内存需求（第 7–9 行）。如果当前可用内存足以移动下一页，我们就立即对其进行调度（第 13–15 行）。在阶段 2 中，我们尽可能提前所有_gather 任务，以便与先前的计算任务重叠。具体而言，对于每个 all_gather 任务，我们逐步尝试将其调度时间提前（即将其触发 ID 减 1）。每次尝试时，我们都会利用跟踪信息、张量大小以及时间前移后的调度方案来衡量峰值内存使用量，从而确定前移后是否会发生内存不足（OOM）错误。此过程会持续进行，直至达到不会发生 OOM 错误的最小触发 ID。最后，将 all_gather 任务连同新的触发 ID 一起重新插入调度方案（第 20–21 行）。这种方法有助于更高效地利用可用内存，并更好地重叠 GPU 上的任务，从而提升性能。

4.3 无锁更新机制

GPU利用率不足。 在训练中引入异构内存或多GPU时，GPU 的计算往往会受到数据传输的阻碍。一方面，GPU 必须等待更新后的参数传入才能进行计算，或者等待计算出的梯度传出以腾出空间。另一方面，GPU 还必须等待通过远程通信完成梯度同步，才能更新本地参数。毫无疑问，这会浪费 GPU 资源，并降低整体效率。更糟的是，在现有系统的训练中引入 SSD 存储会进一步降低 GPU利用率，因为 SSD 的 I/O 速度最慢。以腾讯的 A100 服务器为例，GPU 显存访问、CPU-GPU 传输和 SSD-CPU 传输的 I/O 速度分别为 600 GB/s、32 GB/s 和 3.5 GB/s。我们的观察表明，引入 CPU 内存和 SSD 存储后，近 80% 的迭代时间处于空闲状态；而仅引入 CPU 内存时，这一比例仅为 10%。

为缓解这一训练瓶颈，我们设计了一种新颖的无锁更新机制，通过一种新颖的异步一致性控制协议，将 GPU 计算与 CPU 优化器操作解耦。算法 2 展示了所提出机制的具体细节。其核心思想是在 CPU 内存中使用两个缓冲区，分别存储半精度浮点数参数和梯度，并利用辅助缓冲线程维护这些缓冲区。训练期间，每个 GPU 从 CPU 缓冲区获取半精度浮点数参数，并执行前向和反向计算（第 19–23 行）。随后，生成的梯度被卸载到 CPU 内存（第 24 行），最终由缓冲线程累积到 CPU 缓冲区中（第 15 行）。在优化器

算法 2： 无锁更新机制

输入： $model = \{l_0, \ldots\ldots, l_{n-1}\}$：层列表，其中 l_i是第 i 层
$p_{32}(l_i)$： l_i的 FP32 参数
$m_{32}(l_i)$： l_i的梯度一阶矩（FP32）
$v_{32}(l_i)$： l_i的梯度二阶矩（FP32）
$g_{16}(l_i)$： l_i的梯度（半精度浮点数）
$p'_{16}(l_i)$： l_i 的半精度浮点数缓冲参数
$g'_{16}(l_i)$： l_i 的半精度浮点数缓冲梯度

```
1  /* CPU 上的更新线程 */
2  当 仍有未清除的缓冲梯度时，执行
3      对于 l_i ∈ reverse(model) 执行
4          从 SSD 存储中获取 p32(l_i) m32(l_i) v32(l_i);
5          通过 g'16(l_i) 更新 p32(l_i) m32(l_i) v32(l_i);
6          将 p32(l_i) 传递给缓冲线程;
7          将 p32(l_i) m32(l_i) v32(l_i) 卸载到 SSD 存储;
8
9  /* CPU 上的缓冲线程 */
10 当未完成 时
11     如果 从更新线程收到 p32(l_i)
12         清空缓冲梯度： g'16(l_i) ← 0;
13         更新缓冲参数： p'16(l_i) ← 转换为(p32(l_i)，半精度浮点数);
14     否则，如果收到来自 GPU 的 g16(l_i) 则
15         累积缓冲梯度： g'16(l_i) ← g'16(l_i) + g16(l_i);
16
17 /* GPU 上的计算 */
18 当尚未完成时执行
19     对于 l_i ∈ model 执行
20         从 CPU 内存中获取缓冲参数 p'16(l_i);
21         使用 p'16(l_i) 执行前向计算;
22     对于 l_i ∈ 反向(model) 执行
23         使用 p'16(l_i) 执行反向计算并获取 g16(l_i);
24         将 g16(l_i) 卸载到 CPU 内存（通知缓冲线程）;
```

更新期间，CPU 上的更新线程从固态硬盘读取 FP32 参数和优化器状态，然后根据已缓冲并累积的梯度进行更新（第 4–5 行）。随后，缓冲线程将收到更新缓冲区的通知，包括清除已缓冲的梯度，并将更新后的参数转换为缓冲参数（第 12–13 行）；与此同时，更新后的参数和优化器状态正被卸载到固态硬盘（第 7 行）。

通过引入辅助缓冲区，我们的系统消除了 GPU 计算与 CPU 模型更新之间的锁。虽然这会带来额外的内存开销，但由于缓冲参数和梯度都以半精度浮点数存储，内存占用较小，因此可以接受。无锁机制的另一个副作用是，参数可能会出现陈旧，因为受 SSD I/O带宽限制，CPU 上的更新线程运行速度比 GPU 慢。尽管如此，已有研究证实，深度学习模型训练能够很好地容忍这种陈旧性。在第6节中，我们将通过实验证明，无锁机制在提升效率的同时不会损害收敛性。最后，由于优化器逐元素更新参数，数据传输和 CPU 计算可以按页级别进行调度，从而更好地实现不同资源间的重叠。

```
导入 angelptm

类 MyModel():
    定义 __初始化__(self)
        ...
    定义 forward(self , x)
        ...

模型 = MyModel ()
优化器 = 优化器（模型）
模型 = angelptm . 初始化（模型,
                          优化器,      配置）
...
遍历 批次 在 批次中:
    loss = model(batch)
    model . backward (loss)
    model .step ()
```

图 6：编程接口。

5 实现

Angel-PTM 为工业场景中的高效深度学习模型训练提供了全面的解决方案。它采用了若干关键技术 [30, 33, 38]，基于 Hetu [31], 实现于 PyTorch [40], 之上，并提供页抽象，以提高内存效率；同时采用统一调度方法，提升资源利用率。此外，Angel-PTM 针对 A100 服务器进行了大量优化，使其能够充分发挥硬件能力，以执行深度学习任务。本节将简要介绍关键组件的实现。

追踪器。 Angel-PTM 中的追踪器负责跟踪每个张量的使用情况，并将给定模型的张量访问模式汇总为由以下元素组成的列表：

• *tensor_id*：此张量的逻辑 ID。• *first_id*：首次访问此张量时的逻辑 ID。• *end_id*：最后一次访问此张量时的逻辑 ID。• *cpu_time*：在 CPU 上生成此张量所需的时间。• *gpu_time*：在 GPU 上生成此张量所需的时间。

为了为每个参数分配唯一的张量 ID，我们修改 Parameter 类的 __初始化__ 方法，使其使用全局变量 *id*。随后，我们通过注册钩子函数，跟踪每次迭代中每个参数的首次和最后一次使用，并分别将它们记录为 首次_*id* 和结束_*id*。为了捕获张量在 CPU 和 GPU 上的生成时间，我们分别使用 time.time() 和 CudaEvent 接口，准确测量每个张量在 CPU 和 GPU 上所需的时间。

统一调度器。 统一调度器负责协调分配器、执行器和通信器这三个组件的活动。通过消息传递发送指令会给训练带来严重开销，因此我们采用事件驱动编程技术。例如，只有在修改其输入张量的事件完成后，才会将计算提交到线程中执行。

分配器。Angel-PTM 中的分配器负责在分层内存资源中按页管理张量。它为每个页提供三种内存操作，包括分配、释放和移动。为了降低内存空间申请开销，并利用训练的迭代特性，我们从系统的分层内存中预先分配空间

表 3：硬件环境概览。

一台 GPU 服务器的配置	
CPU CPU 内存	4 ×AMD EPYC 7K62 48 核处理器 32 × 32GB 2933MHz DDR4
GPU GPU 内存	8 ×NVIDIA Tesla A100 Tensor Core GPU 8 × 40GB HBM2
SSD	11TB

，包括 GPU 内存、CPU 固定内存和 SSD 内存。为支持细粒度的内存操作，我们将预分配内存划分为固定大小的页，每个页都可以独立分配、释放和移动。此外，我们分别利用 cudaMemcpyAsync() 和 DeepNVMe 库 [45]实现 GPU-CPU 和 CPU-SSD 数据的异步移动。

执行器。 Angel-PTM 中的执行器负责调度 CPU 和 GPU 等服务器计算设备上的张量计算。同时，它为这些计算设备分别维护一个流，包括 CPU 流和 GPU 流。执行器接收统一调度器发出的指令后，会将计算插入相应的流中，并按照插入顺序将其调度给计算线程。当计算所需的所有输入都准备就绪后，计算便会开始；计算完成后，执行器会向统一调度器反馈。Angel-PTM 中的执行器在动态计算管理机制中发挥着关键作用，该机制可优化不同计算资源使用之间的重叠。

通信器。 Angel-PTM 中的通信器负责调度不同网络设备（包括网卡和 NVLink）之间的通信。我们使用 NCCL 库实现通信器 [21], , NCCL 为多GPU系统提供高效的通信原语。这些原语包括全归约、全收集和归约散播等集合通信操作，这些操作对于训练期间在 GPU 之间交换模型参数和梯度至关重要。通信器还维护一个队列来存储通信任务，并根据统一调度器的指令安排这些任务执行，从而能够重新排列队列中的任务，以提高计算与通信之间的重叠。

在分布式服务器上高效移动数据。GPU 服务器通常具有复杂的互连拓扑，例如 A100 服务器 [2] 包含两颗 CPU、四个 PCIe 交换机和八个 A100 GPU。这些 GPU 可以并行访问 CPU 内存，从而在分布式训练中实现高效的数据移动。为了充分利用这一硬件特性，我们将模型参数均匀划分到各个 GPU 上，以并行移动 CPU 与 GPU 之间的参数，这与 ZeRO-Infinity [45]类似。这进一步加快了训练期间的数据移动，并实现了出色的可扩展性。

6 实验评估

6.1 实验设置

机器环境。我们的实验是在腾讯的生产级 GPU 集群上进行的，每台服务器都

表 4：用于评估的模型。

模型	#层	#注意力头	d_{Model}	d_{FFN}	#专家
GPT3-1.7B	24	24	2304	9216	-
GPT3-13B	40	40	5140	20506	-
GPT3-28B	26	128	8192	32768	-
GPT3-30B	64	36	8192	32768	-
GPT3-55B	68	128	8192	32768	-
GPT3-120B	64	96	12288	49152	-
GPT3-175B	70	112	14336	57344	-
T5-1.4B	16	16	1024	16384	-
T5-27B	28	64	4096	16384	-
T5-58B	60	64	4096	16384	-
T5-MoE-1.2T	16	16	1024	16384	2304

配备 8 块 NVIDIA Tesla A100 GPU，更多详情见表 3。服务器内的 GPU 通过 NVLink-3.0 互连，而服务器之间则通过 RoCE 和 16 16× 网卡连接（总计 12.5 GB/s）。PCIe 带宽为 32 GB/s，固态硬盘峰值带宽为 3.5GB/s。

基准测试。我们对三种大规模 Transformer 模型进行评估，即 GPT-3 [7], T5 [42], 和 T5-MoE [13], ，以验证所提出系统的有效性和可扩展性。为实现不同的模型规模，我们尝试采用不同数量的 Transformer模块、隐藏维度和专家，具体模型配置见表 4。我们使用第 2 节介绍的混合精度技术训练所有这些模型，该技术以 FP32 存储模型状态，同时以 BF16 进行计算。

基线。在 Angel-PTM 之前，DeepSpeed [46] 和 Megatron-LM [35], 因广泛应用，成为腾讯 Taiji（又称 Angel）机器学习平台中采用的两种预训练系统。因此，我们选择这两个系统作为基线，以评估 Angel-PTM 的有效性。DeepSpeed 是一种异构训练解决方案，目前实现了业界领先的性能；为确保公平比较，我们使用其官方示例。对于 Megatron-LM，我们手动为每个实验模型搜索最佳并行策略，最终得到一种混合并行解决方案，将数据并行、模型并行和流水线并行结合起来。

我们开展了一系列实验，以展示 Angel-PTM 在训练大规模 Transformer 模型方面的有效性和可扩展性。具体而言，我们在第 6.2 节评估系统在单台服务器上支持的最大模型规模，并在第 6.3 节比较两种 GPU 配置下不同模型的吞吐量。此外，我们在第 6.4 节展示了系统使用数百个 GPU 训练十亿级稠密模型和万亿级稀疏模型时的可扩展性。进一步地，我们在第 6.5 节评估所提出的无锁更新机制的收敛性能，以进一步验证在引入 SSD 存储、将模型规模扩展到极限时系统的有效性。值得注意的是，除使用 SSD 存储进行训练的第 6.5 节实验外，其他各节默认使用 CPU 和 GPU 的内存。

6.2 模型规模

我们首先在单台服务器上进行评估，以测试 Angel-PTM 和 DeepSpeed 所能支持的最大模型规模及相应的最大吞吐量，这些评估结果可以通过逐步增加 Transformer模块的数量并固定其他模型设置获得。两个系统都使用 ZeRO-3 [44], 将模型均匀地划分到多个GPU上，从而实现模型规模的高效线性扩展。

表 5：单台服务器支持的最大模型规模。

模型	系统	#参数	#批次	GPU 内存	样本数/秒
GPT	DeepSpeed	28B	1	18	0.404
		28B	36	40	7.61
	AngelPTM	28B	38	39	**10.99**
		55B	1	33	0.464
		55B	10	40	3.34
T5	DeepSpeed	27B	1	20	0.317
		27B	32	39	7.31
	AngelPTM	27B	50	40	**14.38**
		58B	1	38	0.432
		58B	4	40	3.37

结果汇总于表 5。对于 GPT 模型，我们将注意力头数量设为 128，嵌入维度设为 8192，前馈神经网络隐藏层大小设为 32768。DeepSpeed 最多支持 28B 的模型规模，共 26 层；而 Angel-PTM 可进一步扩展至 55B，共 68 层，提升了 96.4 %。需要注意的是，尽管每个 GPU 有 22GB 可用内存，DeepSpeed 仍无法扩展到更大的模型规模。这是因为 DeepSpeed 会静态地将模型状态分配到 GPU 和 CPU 上，因此最大模型规模受 CPU 内存限制。相比之下，为了充分利用这些可用内存，Angel-PTM 采用动态内存管理，将部分模型状态移入 GPU 内存，从而支持更大的模型规模。关于训练效率，具体而言，表 5 中的样本数/秒一列显示，DeepSpeed 的最大吞吐量为 7.61 样本数/秒，而 Angel-PTM 为 10.99 样本数/秒，提升了 44%。此外，Angel-PTM 对其支持的最大 GPT 模型（55B）的训练效率为 3.34 样本数/秒。这些分析也适用于 T5 模型：Angel-PTM 的最大模型规模提升了 114.8%，吞吐性能提升了 96.7%。

总而言之，与 DeepSpeed 相比，Angel-PTM 能够：(1) 在使用相同硬件资源的情况下支持更大规模的模型；(2) 对于相同的模型实现更高的训练效率。

6.3 吞吐量

为了验证 Angel-PTM 的训练效率，我们评估了每个竞品的吞吐量。具体而言，我们分别在 8 个 GPU 和 32 个 GPU 上，以最大批量大小训练了一系列 GPT 模型，GPT 模型的规模范围为 1.7 B 至 120B。为了清晰地比较各系统，我们以 DeepSpeed 为基准对每个系统的吞吐量进行归一化。结果如图 7 所示。

在 1x8 GPU 的配置下，除 1.7B 模型外，Angel-PTM 在训练效率方面始终优于其他系统。这是因为 1.7B 模型足够小，可以由单个 GPU 容纳，因此，原生数据并行（不使用 ZeRO）能实现最佳性能，Megatron-LM 也采用了这一策略。由于 Angel-PTM 即使在无需移动内存

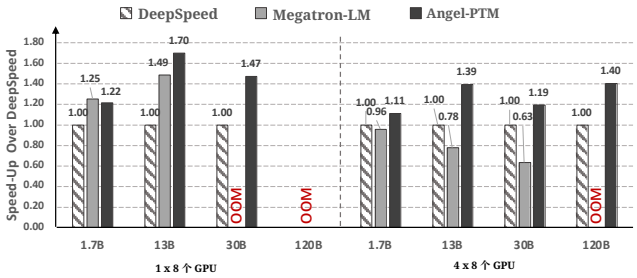

图 7：将 Angel-PTM 与其他知名预训练系统在一系列 GPT 模型上的表现进行比较。

的情况下也会产生额外的内存管理开销，因此运行速度略低于 Megatron-LM （慢 2.4% ）。对于 1.7B 和 13B 模型，Angel-PTM 和 Megatron-LM 的表现都优于 DeepSpeed。这是合理的，因为 DeepSpeed 会在 CPU 内存和 GPU 内存之间静态划分模型状态，导致冗余的内存移动。然而，当模型规模增至 30B 时，Megatron-LM 因 GPU 内存有限而出现内存溢出错误；得益于基于细粒度生命周期的调度方法，Angel-PTM 仍优于 DeepSpeed。该方法按需将模型状态划分到 CPU，并在恰当的时机调度其移动。

在 4x8 GPU 的配置下，Angel-PTM 的性能依然最佳。使用更多 GPU 时，Megatron-LM 能够支持 30B 模型，而 DeepSpeed 和 Angel-PTM 借助 ZeRO 技术则能够进一步支持 120B 模型。Angel-PTM 和 DeepSpeed 的表现都优于 Megatron-LM，因为它们能够使用更大的微批次大小进行训练。

总之，实验结果表明，在吞吐性能方面，Angel-PTM 始终优于 DeepSpeed 和 Megatron-LM。具体而言，Angel-PTM 的表现优于最先进的分层训练系统 DeepSpeed，平均提升.4%，最高提升 70%。此外，与最先进的混合并行训练系统 Megatron-LM 相比，Angel-PTM 平均提升.9%，最高提升.9%。

6.4 可扩展性

为验证 Angel-PTM 的可扩展性，我们对两个超大规模 Transformer模型进行了评估，分别是 GPT3-175B 和 T5-MoE-1.2T。

对 GPT3-175B 的评估。GPT3-175B 模型最初由 OpenAI [7], 提出，该模型使用海量文本数据进行训练，并能完成多种自然语言处理任务，例如语言翻译和问答。它还是 ChatGPT [1], 的基础模型，而 ChatGPT 是人工智能领域的一项突破性成果。因此，验证我们的系统支持该模型的能力和可扩展性至关重要。

GPT-175B 的配置详见表 4，我们在图 8 中展示了使用不同数量的 GPU 训练该模型时的吞吐量。结果表明，在训练 GPT-175B 模型时，Angel-PTM 实现了超线性可扩展性。

我们观察到，在 32 个节点（256 个 GPU）上，吞吐量为 11.68 样本数/秒；在 96 个节点（768 个 GPU）上，吞吐量提升至 36.46 样本数/秒，实现了 3.12× 加速。随着 GPU 数量增加，模型

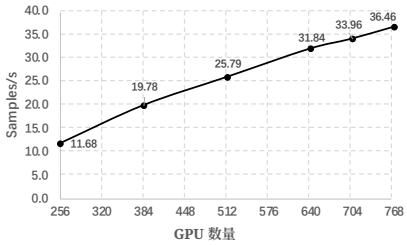

图 8：训练 GPT3 模型的可扩展性。

状态会分布到更多 GPU 上。这使我们能够增大全局批次大小，从而充分利用可用的 GPU 内存。此外，优化器更新过程会在更多 CPU 之间并行执行，数据移动也会在更多 PCIe 之间并行进行。这些因素促成了超线性可扩展性和更高的训练吞吐量。

结果表明，我们的系统能够充分利用不断增加的 GPU 数量，并在 PCIe 上高效地并行处理数据移动，这对于及时且经济高效地训练大型模型至关重要。

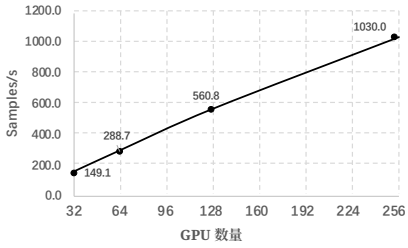

图 9：训练 T5-MoE 模型的可扩展性。

对 T5-MoE-1.2T 的评估。T5-MoE 模型是另一类知名的大型模型，采用稀疏混合专家架构，最初由谷歌提出，名称为 Switch-Transformer [13, 27]。混合专家模型可以通过将输入数据路由到少量专家网络，大幅降低训练成本。Angel-PTM 使用专家并行 [37],训练 T5-MoE 模型，其中混合专家层内的专家参数分片到所有 GPU 上，而非混合专家参数则进行复制。

T5-MoE-1.2T 模型的每个 MoE 层包含 2304 个专家，并将每个 GPU 上每个 MoE 层的专家数固定为 9，以便在改变 GPU 数量时实现不同的模型规模。例如，在 128 个 GPU 上训练的 T5-MoE 模型，每个 MoE 层包含 1152 个专家。详细配置见表 4。我们在图 9 中总结了结果，表明 Angel-PTM 在训练 T5-MoE 模型时具有近线性可扩展性。随着更多服务器参与训练，输入到 MoE 层全互连通信中的数据会增多，这可能导致吞吐量下降。因此，T5-MoE-1.2T 的可扩展性低于 GPT3-175B。

表 6：使用固态硬盘训练大型 T5-MoE 模型的评估。

系统	#参数	#GPU	样本数/秒	有效损失↓
AngelPTM	1T	64	37.26	1.124
	10T	576	317.82	**0.853**
+ 无锁	10T	576	**942.31**	0.861

6.5 推进对极端模型规模的支持

正如 Kaplan 等人所指出的， [25],较大的模型往往优于较小的模型，这促使许多研究人员不断增大模型规模。在这项研究中，我们使用参数量达 10T 的放大版 T5-MoE 模型，评估 Angel-PTM 对超大规模模型的支持能力。

我们基于工业文本数据集，对 T5-MoE-1T 和 T5-MoE-10T 进行预训练。为了支持极端模型规模，我们将 SSD 存储引入训练过程。实验结果如表 6 所示，其中包括以样本数/秒表示的训练效率，以及以验证损失衡量的模型质量。在 T5-MoE-1T 模型上，Angel-PTM 使用 64 个 GPU 实现了 37.26 样本数/秒的吞吐量。当我们通过增加专家数量将模型规模扩大到 10T 时，Angel-PTM 使用 64 个 GPU 实现了 317.72 样本数/秒的吞吐量，展现出近线性可扩展性。

此外，在引入 SSD训练时，SSD 的 I/O带宽会显著影响 CPU参数更新的速度，从而降低整体训练速度。通过利用第6节中的无锁更新机制 ，在 CPU 和 GPU 之间执行异步更新，Angel-PTM 显著提升了整体训练吞吐量。具体而言，对于 T5-MoE-10T 模型，启用无锁机制后，训练吞吐量从 317.82 样本数/秒提升至 962.31 样本数/秒，加速比达到 2.96×。与此同时，验证损失的实验结果表明，该机制对模型质量影响很小。

7 相关工作

分布式训练系统。许多知名系统已被设计并实现，用于大规模模型训练。DeepSpeed 提出了针对数据并行的 ZeRO 优化，它根据不同的优化级别，将模型状态均匀分区到所有设备上：第 1 阶段分区优化器状态（例如，32 位参数以及 Adam 的一阶和二阶矩 [26]），第 2 阶段分区梯度，第 3 [43] 阶段分区 16 位参数。然而，这种方法需要额外通信才能获得完全更新的模型参数，因此，尽管优化级别越高可节省越多内存，但也会带来更大的时间开销。Megatron-LM 提出了一种名为张量并行的新方法，它易于实现，并且在 A100 等通过 NVLink-3.0 [35] 高度互连的 GPU 上执行效率很高。Galvatron [32] 提出了一种高效算法，可找到将数据并行、模型并行、流水线并行和 ZeRO 优化结合起来的最优策略，从而高效训练大型模型。由于 GPU 内存有限，ZeRO 无法用于扩展大型模型，而模型并行包括 Megatron-LM 和 Galvatron 在内的技术，由于复杂性较高，难以在工业环境中部署。Angel-PTM 通过将 ZeRO 与分层内存相结合来应对这些挑战，从而兼具灵活性和良好的可扩展性。

异构训练系统。 随着深度学习的发展，训练模型的类型也从早期的基于 CNN 的模型 [18]演变为当前的基于 Transformer 的模型 [7, 9, 41, 42]。对于基于 CNN 的模型，内存利用率主要受激活值占用，研究主要聚焦于优化单 GPU 内存。Wang 等人 [53] 提出根据成本分析，通过重计算或将部分激活值卸载到 CPU 来释放内存。Nie 等人 [38] 提出了张量切分优化方法，以实现细粒度内存管理，从而帮助突破内存瓶颈并制定更高效的执行计划。基于 Transformer 的模型由于模型状态规模庞大而具有较高的内存利用率，因此相关研究致力于优化分布式训练中的 GPU 内存。Ren 等人 [46]提出将优化器计算和模型状态卸载到 CPU，以节省 GPU 内存，并进一步将其与独特的最优卸载策略以及由 ZeRO 支持的数据并行相结合。Rajbhandari 等人 [45] 将固态硬盘存储引入训练，并提出带宽中心化分区算法，将模型分布到所有设备上。Fang 等人 [12] 通过基于块的内存管理器，在训练过程中动态管理模型状态。与现有系统不同，Angel-PTM 通过 页 实施细粒度内存管理，以减少内存碎片并提高训练效率。

GPU 内存管理。 已有许多内存管理优化方案，通过预先分配大部分 GPU 内存并自行管理，包括分页、替换缓存和内存池。Mosaic 提供了对多种页面大小的应用透明支持，以实现换入和换出。MultiQx-GPU 为 GPU 数据库中的并发查询设计了由成本驱动的替换策略，以提升执行效率。Zhang 等人提出了内存池 *CNMeM*，利用生命周期语义减少内存碎片，并设计了一种启发式算法来简化优化问题。然而，关于分页、替换缓存和统一内存地址的研究并非针对深度学习训练而设计，也未利用张量访问模式的特殊性；而其他关于内存池的研究则未考虑 CPU 内存。Angel-PTM 利用生命周期信息提升整体性能，减少了内存碎片，并改善了不同资源之间的重叠。

8 结论

本文介绍了 Angel-PTM，这是腾讯开发的一款易于使用且高效的深度学习系统，适用于预训练和微调任务。为了便于用户使用并实现无缝扩展，我们以数据并行、参数分片和分层内存为基础设计了 Angel-PTM。为充分利用训练期间的内存和带宽，我们引入了 页抽象，以实现细粒度内存管理，并提出了一种统一调度方法，用于全面管理训练期间的关键操作。此外，我们集成了 SSD 存储，使预训练模型能够扩展到极大规模，并开发了

一种无锁更新机制，以解决 SSD I/O 带宽瓶颈。实证结果表明，在最大支持模型规模和训练吞吐量方面，Angel-PTM 均优于现有系统。

参考文献

[1] 2022. ChatGPT：面向对话优化语言模型。https://openai.com/ blog/chatgpt/.

[2] 2022. DGX A100 系统整体架构。https://www.microway.com/hpc-tech-tips/dgx-a100-review-throughput-and-hardware-summary/.[3] Martín Abadi、Paul Barham、Jianmin Chen、Zhifeng Chen、Andy Davis、Jeffrey Dean、Matthieu Devin、Sanjay Ghemawat、Geoffrey Irving、Michael Isard 等。2016。TensorFlow：一个用于大规模机器学习的系统。发表于*OSDI*，第 16 卷。美国佐治亚州萨凡纳，265–283 页。[4] Rachata Ausavarungnirun、Joshua Landgraf、Vance Miller、Saugata Ghose、Jayneel Gandhi、Christopher J. Rossbach 和 Onur Mutlu。2017年。Mosaic：一款支持多种页面大小且对应用透明的 GPU 内存管理器。发表于*MICRO*。 136 150 10 1145 3123939 3123975 –。

[5] Lei Jimmy Ba、Jamie Ryan Kiros 和 Geoffrey E. Hinton。2016。层归一化。*CoRR* abs/1607.06450（2016）。arXiv:1607.06450 http://arxiv.org/abs/1607. 06450[6] Rishi Bommasani、Drew A Hudson、Ehsan Adeli、Russ Altman、Simran Arora、Sydney von Arx、Michael S Bernstein、Jeannette Bohg、Antoine Bosselut、Emma Brunskill 等。2021年。基础模型的机遇与风险。*arXiv* 预印本 *arXiv:2108.07258*（2021）。[7] Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。2020年。语言模型是少样本学习器。神经信息处理系统进展 33（2020），1877–1901。[8] Tianqi Chen、Bing Xu、Chiyuan Zhang 和 Carlos Guestrin。2016。以次线性内存开销训练深度网络。*arXiv* 预印本 *arXiv:1604.06174*（2016）。[9] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。2019年。BERT：用于语言理解的深度双向Transformer预训练。发表于*2019*年北美计算语言学协会分会年会论文集：人类语言技术，第 *1* 卷（长短论文）。4171–4186 页。[10] Linhao Dong、Shuang Xu 和 Bo Xu。2018年。Speech-transformer：一种用于语音识别的无循环序列到序列模型。发表于*2018*年 *IEEE* 国际声学、语音与信号处理会议（*ICASSP*）。IEEE，5884–5888 页。[11] Alexey Dosovitskiy、Lucas Beyer、Alexander Kolesnikov、Dirk Weissenborn、Xiaohua Zhai、Thomas Unterthiner、Mostafa Dehghani、Matthias Minderer、Georg Heigold、Sylvain Gelly、Jakob Uszkoreit 和 Neil Houlsby。2021年。一张图像胜过 16×16 个词：用于图像识别的Transformer及其缩放。发表于国际学习表征会议。[12] Jiarui Fang、Zilin Zhu、Shenggui Li、Hui Su、Yang Yu、Jie Zhou 和 Yang You。2022年。通过基于块的动态内存管理并行训练预训练模型。*IEEE* 并行与分布式系统汇刊 34，1（2022），304–315。[13] William Fedus、Barret Zoph 和 Noam Shazeer。2022年。Switch Transformers：通过简单高效的稀疏性缩放至万亿参数模型。机器学习研究期刊 23，120（2022），1–39。[14] Fangcheng Fu、Yuzheng Hu、Yihan He、Jiawei Jiang、Yingxia Shao、Ce Zhang 和 Bin Cui。2020年。别浪费你的比特！通过 TinyScript 压缩深度神经网络的激活值和梯度。发表于第 *37* 届国际机器学习大会论文集，*ICML 2020，2020*年*7*月*13*日至*18*日，线上会议，第 119 卷。3304–3314 页。[15] Fangcheng Fu、Xupeng Miao、Jiawei Jiang、Huanran Xue 和 Bin Cui。2022年。通过启用缓存的本地更新，实现通信高效的纵向联邦学习训练。*VLDB*基金会会刊 15，10（2022），2111–2120。[16] Fangcheng Fu、Yingxia Shao、Lele Yu、Jiawei Jiang、Huanran Xue、Yangyu Tao 和 Bin Cui。2021年。VF^{2}Boost：用于跨企业学习的超快速纵向联邦梯度提升。发表于*SIGMOD '21*：国际数据管理会议，线上会议，中国，*2021*年*6*月*20*日至*25*日。563–576 页。[17] Amir Gholami、Zhewei Yao、Sehoon Kim、Michael W Mahoney 和 Kurt Keutzer。2021年。人工智能与内存墙。[18] Kaiming He、Xiangyu Zhang、Shaoqing Ren 和 Jian Sun。2016。用于图像识别的深度残差学习。发表于*IEEE* 计算机视觉与模式识别会议论文集。770–778 页。[19] Dan Hendrycks 和 Kevin Gimpel。2016。高斯误差线性单元（GeLU）。*arXiv* 预印本 *arXiv:1606.08415*（2016）。[20] Yanping Huang、Youlong Cheng、Ankur Bapna、Orhan Firat、Dehao Chen、Mia Chen、HyoukJoong Lee、Jiquan Ngiam、Quoc V Le、Yonghui Wu 等。2019年。GPipe：利用流水线并行高效训练巨型神经网络。神经信息处理系统进展 32（2019），103–112。[21] Sylvain Jeaugey。2017年。NCCL 2.0。发表于*GPU*技术大会（*GTC*），第 2 卷。

[22] Jiawei Jiang、Fangcheng Fu、Tong Yang 和 Bin Cui。2018年。SketchML：利用数据摘要加速分布式机器学习。载于《2018年国际数据管理会议论文集》，SIGMOD 2018会议，休斯顿，美国得克萨斯州，2018年6月10日至15日。1269–1284。[23] Jiawei Jiang、Pin Xiao、Lele Yu、Xiaosen Li、Jiefeng Cheng、Xupeng Miao、Zhipeng Zhang 和 Bin Cui。2020年。PSGraph：腾讯如何使用 Spark 训练超大规模图？载于《2020年 IEEE第36届数据工程国际会议》（ICDE）。IEEE，1549–1557。[24] Jie Jiang、Lele Yu、Jiawei Jiang、Yuhong Liu 和 Bin Cui。2018年。Angel：一种新型大规模机器学习系统。《国家科学评论》5卷2期（2018年），216–236。[25] Jared Kaplan、Sam McCandlish、Tom Henighan、Tom B Brown、Benjamin Chess、Rewon Child、Scott Gray、Alec Radford、Jeffrey Wu 和 Dario Amodei。2020年。神经语言模型的缩放定律。CoRR abs/2001.08361（2020年）。[26] Diederik P. Kingma 和 Jimmy Ba。2015年。Adam：一种随机优化方法。载于《国际学习表征会议》。[27] Dmitry Lepikhin、HyoukJoong Lee、Yuanzhong Xu、Dehao Chen、Orhan Firat、Yanping Huang、Maxim Krikun、Noam Shazeer 和 Zhifeng Chen。2021年。GShard：利用条件计算和自动分片扩展巨型模型。载于《国际学习表征会议》。[28] Mu Li、David G Andersen、Jun Woo Park、Alexander J Smola、Amr Ahmed、Vanja Josifovski、James Long、Eugene J Shekita 和 Bor-Yiing Su。2014年。利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）。583–598。[29] Ze Liu、Yutong Lin、Yue Cao、Han Hu、Yixuan Wei、Zheng Zhang、Stephen Lin 和 Baining Guo。2021年。Swin Transformer：采用移位窗口的层次化视觉Transformer。载于《IEEE/CVF计算机视觉国际会议论文集》。10012–10022。[30] Xupeng Miao、Xiaonan Nie、Yingxia Shao、Zhi Yang、Jiawei Jiang、Lingxiao Ma 和 Bin Cui。2021年。通过部分归约实现异构性感知分布式机器学习训练。载于《SIGMOD会议论文集》。ACM，2262–2270。https: //doi.org/ 10 1145 3448016 3452773 . / .

[31] Xupeng Miao、Xiaonan Nie、Hailin Zhang、Tong Zhao 和 Bin Cui。2023年。Hetu：一种高效的自动并行分布式深度学习系统。《中国科学：信息科学》66卷1期（2023年），1–2。[32] Xupeng Miao、Yujie Wang、Youhe Jiang、Chunan Shi、Xiaonan Nie、Hailin Zhang 和 Bin Cui。2023年。Galvatron：利用自动并行在多GPU上高效训练Transformer。VLDB基金会会刊16卷3期（2023年），470–479。https://doi.org/10.14778/3570690.3570697[33] Xupeng Miao、Hailin Zhang、Yining Shi、Xiaonan Nie、Zhi Yang、Yangyu Tao 和 Bin Cui。2021年。HET：通过支持缓存的分布式框架扩展超大嵌入模型训练。VLDB基金会会刊15卷2期（2021年），312–320。[34] Paulius Micikevicius、Sharan Narang、Jonah Alben、Gregory Diamos、Erich Elsen、David Garcia、Boris Ginsburg、Michael Houston、Oleksii Kuchaiev、Ganesh Venkatesh 和 Hao Wu。2018年。混合精度训练。载于《国际学习表征会议》。[35] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Korthikanti、Dmitri Vainbrand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro 等。2021年。使用 Megatron-LM 在GPU集群上高效训练大规模语言模型。载于《高性能计算、网络、存储与分析国际会议论文集》。1–15。[36] Xiaonan Nie、Xupeng Miao、Shijie Cao、Lingxiao Ma、Qibin Liu、Jilong Xue、Youshan Miao、Yi Liu、Zhi Yang 和 Bin Cui。2021年。Evomoe：一种通过稠密到稀疏门控实现的演进式混合专家训练框架。arXiv预印本 arXiv:2112.14397（2021年）。[37] Xiaonan Nie、Xupeng Miao、Zilong Wang、Jilong Xue、Lingxiao Ma、Zichao Yang、Gang Cao 和 Bin Cui。2023年。FlexMoE：通过动态设备放置扩展大规模稀疏预训练模型训练。SIGMOD会议论文集（2023年）。[38] Xiaonan Nie、Xupeng Miao、Zhi Yang 和 Bin Cui。2022年。TSPLIT：通过张量切分实现高效DNN训练的细粒度GPU内存管理。载于《2022年IEEE第38届数据工程国际会议》（ICDE）。IEEE，2615–2628。[39] Long Ouyang、Jeff Wu、Xu Jiang、Diogo Almeida、Carroll L Wainwright、Pamela Mishkin、Chong Zhang、Sandhini Agarwal、Katarina Slama、Alex Ray 等。2022年。通过人类反馈训练语言模型遵循指令。arXiv预印本 arXiv:2203.02155（2022年）。[40] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。2019年。PyTorch：一种命令式风格的高性能深度学习库。《神经信息处理系统进展》32卷（2019年）。[41] Alec Radford、Jeffrey Wu、Rewon Child、David Luan、Dario Amodei 和 Ilya Sutskever。2019年。语言模型是无监督多任务学习器。OpenAI博客1卷8期（2019年），9。[42] Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li 和 Peter J Liu。2020年。探索统一文本到文本Transformer的迁移学习极限。《机器学习研究期刊》

LearningResearch 21卷（2020年），1–67。[43] Samyam Rajbhandari、Conglong Li、Zhewei Yao、Minjia Zhang、Reza Yazdani Aminabadi、Ammar Ahmad Awan、Jeff Rasley 和 Yuxiong He。2022年。DeepSpeed-MoE：推进混合专家推理与训练，助力下一代人工智能规模化。载于国际机器学习大会。PMLR，18332–18346。[44] Samyam Rajbhandari、Jeff Rasley、Olatunji Ruwase 和 Yuxiong He。2020年。ZeRO：面向万亿参数模型训练的内存优化。载于*SC20*：高性能计算、网络、存储与分析国际会议。IEEE，1–16。[45] Samyam Rajbhandari、Olatunji Ruwase、Jeff Rasley、Shaden Smith 和 Yuxiong He。2021年。ZeRO-Infinity：突破极大规模深度学习的GPU内存墙。载于高性能计算、网络、存储与分析国际会议论文集。1–14。[46] Jie Ren、Samyam Rajbhandari、Reza Yazdani Aminabadi、Olatunji Ruwase、Shuangyan Yang、Minjia Zhang、Dong Li 和 Yuxiong He。2021年。ZeRO-Offload：让数十亿规模的模型训练普及化。载于*USENIX*年度技术会议。551–564。[47] Carlos Riquelme、Joan Puigcerver、Basil Mustafa、Maxim Neumann、Rodolphe Jenatton、André Susano Pinto、Daniel Keysers 和 Neil Houlsby。2021年。利用稀疏混合专家扩展视觉。《神经信息处理系统进展》34卷（2021年），8583–8595。[48] Robin Rombach、Andreas Blattmann、Dominik Lorenz、Patrick Esser 和 Björn Ommer。2021年。利用潜在扩散模型进行高分辨率图像合成。arXiv:2112.10752 [cs.CV][49] Shaden Smith、Mostofa Patwary、Brandon Norick、Patrick LeGresley、Samyam Rajbhandari、Jared Casper、Zhun Liu、Shrimai Prabhumoye、George Zerveas、Vijay Korthikanti 等。2022年。使用DeepSpeed和Megatron训练Megatron-Turing NLG 530B：一个大规模生成式语言模型。*arXiv*预印本 *arXiv:2201.11990*（2022年）。

[50] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Lukasz Kaiser 和 Illia Polosukhin。2017年。Attention is All you Need。发表于*NeurIPS*。5998–6008。[51] Guibin Wang、YiSong Lin 和 Wei Yi。2010年。内核融合：一种提升多线程 GPU 能效的有效方法。发表于*2010*年 *IEEE/ACM* 国际绿色计算与通信会议暨国际网络、物理与社会计算会议。IEEE，344–350。[52] Kaibo Wang、Kai Zhang、Yuan Yuan、Siyuan Ma、Rubao Lee、Xiaoning Ding 和 Xiaodong Zhang。2014年。基于 GPU 的并发分析查询处理。*VLDB*基金会会刊 7，11（2014年），1011–1022。https://doi.org/10.14778/2732967.2732976[53] Linnan Wang、Jinmian Ye、Yiyang Zhao、Wei Wu、Ang Li、Shuaiwen Leon Song、Zenglin Xu 和 Tim Kraska。2018年。Superneurons：用于训练深度神经网络的动态 GPU 内存管理。发表于第*23*届 *ACM SIGPLAN* 并行编程原理与实践研讨会论文集。41–53。[54] Yongqiang Wang、Abdelrahman Mohamed、Due Le、Chunxi Liu、Alex Xiao、Jay Mahadeokar、Hongzhao Huang、Andros Tjandra、Xiaohui Zhang、Frank Zhang 等。2020年。用于混合式语音识别的基于 Transformer 的声学建模。发表于 *ICASSP 2020—2020 IEEE* 国际声学、语音与信号处理会议（*ICASSP*）。IEEE，6874–6878。[55] Xu Xie、Fei Sun、Zhaoyang Liu、Shiwen Wu、Jinyang Gao、Jiandong Zhang、Bolin Ding 和 Bin Cui。2022年。面向序列推荐的对比学习。发表于第*38*届 *IEEE* 国际数据工程会议（*ICDE 2022*），马来西亚吉隆坡，*2022*年*5*月*9*日至*12*日。IEEE，1259–1273。https://doi.org/10.1109/ ICDE53745.2022.00099[56] Junzhe Zhang、Sai-Ho Yeung、Yao Shu、Bingsheng He 和 Wei Wang。2019年。基于 GPU 的深度学习系统的高效内存管理。*CoRR* abs/ 1903 06631 2019 1903 06631 。（）。arXiv：。Zhipeng Zhang、Bin Cui、Yingxia Shao、Lele Yu、Jiawei Jiang 和 Xupeng Miao。2019年。Ps2：Spark 上的参数服务器。发表于*2019*年国际数据管理会议论文集。376–388。[57]