一种面向多模态模型的分区混合并行优化方法

巨涛，丁肖健，郭东雨，火久元
（兰州交通大学电子与信息工程学院，730070，兰州）

摘要：针对多模态模型架构复杂、参数量庞大、算力需求高，导致训练难度大、效率低，以及现有数据并行与模型并行策略难以应对内部异构特性的问题，提出了一种面向多模态模型的分区混合并行优化(MMHP)方法。首先根据多模态模型中不同子模块的异构性，构建模块依赖图，识别关键切分点，实现模块分区，保证子模块间的负载均衡；其次，针对不同模块分区的参数规模和计算需求，融合数据并行与模型并行，设计混合并行优化方法以适应异构计算任务的多样化需求；最后，基于动态规划设计计算任务调度优化算法，动态分配计算资源，实现计算任务与资源合理匹配，进一步优化计算资源的利用率，提升模型的训练效率。实验结果表明，在不影响模型训练精度的情况下，所提出的 MMHP 方法可充分利用计算资源，提升多模态模型的训练效率，与现有主流并行策略相比，训练速度最大可提升 2 倍。

关键词：多模态模型；混合并行；并行优化；模块分区；调度优化
中图分类号：TP311 **文献标志码**：A
DOI：10.7652/xjtuxb202602022 **文章编号**：0253-987X(2026)02-0229-12

A Partitioned Hybrid Parallel Optimization Method for Multimodal Models

JU Tao, DING Xiaojian, GUO Dongyu, HUO Jiuyuan
(School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China)

Abstract: To address the challenges of complex architecture, large parameter size, and high computational demands in multimodal models, which lead to difficult training and low efficiency, as well as the limitations of existing data and model parallelism strategies in handling internal heterogeneity, a partitioned hybrid parallel optimization (MMHP) method for multimodal models is proposed. First, based on the heterogeneity of different submodules in the multimodal model, a module dependency graph is constructed to identify key partitioning points, achieving module partitioning and ensuring load balancing among submodules. Second, according to the parameter scale and computational requirements of different module partitions, a hybrid parallel optimization method is designed by integrating data parallelism and model parallelism to accommodate the diverse needs of heterogeneous computing tasks. Finally, a computational task scheduling optimization algorithm is developed based on dynamic programming to dynamically allocate computing resources, achieving a reasonable match between computational tasks and resources, further optimizing the utilization of computing resources, and improving model training efficiency. Experimental results show that, without compromising model training accuracy, the proposed

收稿日期：2025-04-14。 作者简介：巨涛(1980—)，男，教授，硕士生导师。 基金项目：国家自然科学基金资助项目(62262038)；兰州市科技计划资助项目(2025-2-41)。
网络出版时间：2025-07-28 网络出版地址：https://link.cnki.net/urlid/61.1069.t.20250728.1428.004

MMHP method can fully utilize computing resources and improve the training efficiency of multimodal models. Compared with existing mainstream parallel strategies, the training speed can be increased by up to 2 times.

Keywords: multimodal models; hybrid parallel; parallel optimization; module partitioning; scheduling optimization

多模态模型因其跨模态协同能力已成为人工智能研究新热点[1-3]。与单模态语言模型如生成式预训练变换器(GPT)[4]和 Meta AI 公司的大语言模型(LLaMA)[5]相比,多模态模型由于要处理多种模态数据,模型结构和训练过程更加复杂,计算更加密集。一方面,多模态模型由多个异构子模块组成,这些子模块在参数规模、计算需求和任务特性上存在显著差异;另一方面,模型参数规模与数据量呈指数级增长,导致计算资源消耗激增、存储瓶颈日益凸显,训练周期进一步延长,单个图形处理器(GPU)设备有限内存容量和算力已完全无法满足大规模多模态模型的训练需求,这使得多模态模型训练面临巨大挑战[6-9]。

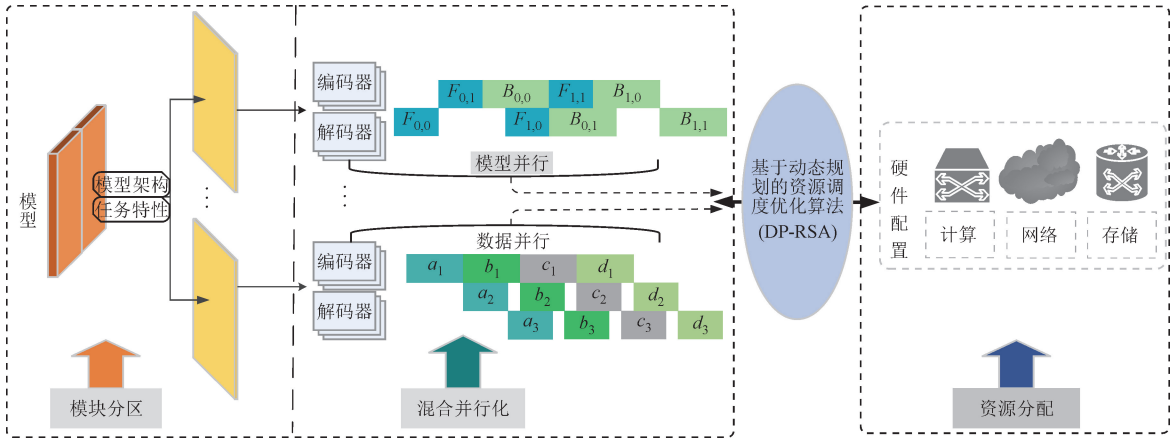
为满足多模态模型训练的高需求,分布式并行训练成为突破计算与存储瓶颈的关键手段,当前主流的并行策略主要包括数据并行和模型并行,模型并行又细分为张量并行与流水线并行[10-12]。文献[13]提出的 UNITER 模型采用数据并行策略,通过训练数据分片与梯度同步实现加速,但该方法的显存占用随模型参数规模呈线性增长,难以适应大规模参数场景下的扩展需求。相比之下,模型并行通过参数切分降低单设备显存压力。文献[14]提出的 PipeWeaver 框架利用流水线机制将模型计算分阶段执行,有效缓解了单设备显存压力,但当前流水线并行方法存在结构性约束,其性能高度依赖模型图的拓扑规则性,对于存在复杂跨层依赖或非顺序拓扑的模型,阶段划分与调度将面临通信开销增加、流水线气泡比例上升及负载均衡困难等挑战[15]。在张量并行方面,文献[16]的 Flamingo 模型通过层内参数切分降低显存压力,利用权重矩阵分解实现分布式训练,但其通信开销与切片粒度密切相关,当参数切片粒度过小时,通信频率上升,导致通信成本激增,同时张量并行的时间复杂度较高,对高速网络带宽要求严格,若网络带宽受限,训练效率将下降[17]。为进一步提升集群资源利用率,文献[18]提出混合并行策略,融合数据并行与模型并行的优势,但该方法未充分考虑多模态模型内部的子模态依赖关系和异构性,导致模型切分不合理,引入冗余计算和通信量,造成计算资源浪费[19-21]。此外,传统静态资源调度策略难以适应多模态模型复杂的计算需求,进一步制约了多模态模型训练效率的提升[22-23]。

综上,现有并行方案在异构模块协同与动态资源适配层面仍存在优化空间,针对上述问题,提出一种面向多模态模型的分区混合并行化方法。该方法首先深入分析多模态模型中各子模块的异构特性,构建模块依赖关系图,识别影响负载分布和通信效率的关键切分点,从而优化模块划分,减少因负载不均衡带来的额外通信开销;其次,针对分区后的子模块,依据其参数规模、计算需求和任务特性,结合数据并行与模型并行设计自适应并行策略,以高效适配多样化的异构计算任务;最后,基于动态规划思想设计任务调度优化算法,实现计算任务与资源的合理映射,提升资源利用效率,减少资源浪费与等待时间。

1 系统框架

针对多模态模型在分布式训练中的挑战,本文提出了一种面向多模态模型的分区混合并行优化方法,旨在充分挖掘异构计算资源的潜力,优化复杂模型的分布式训练过程,解决模型结构复杂、任务负载不均衡以及资源利用率低等问题。所提方法整体框架由模块分区与资源分配两大核心部分组成,两者相互协同,通过合理划分模型结构与高效调度系统资源,实现训练过程的优化与加速,具体系统框架如图 1 所示。

在模块分区方面,针对多模态模型的异构特征,通过深入分析模型内部各子模块间的依赖关系,构建模块依赖关系图,精准识别关键切分点,实现高效的模块划分。在模块划分过程中,充分考虑各子模块的不同计算异构性,任务负载和并行性,以确保负载均衡,避免引入额外的通信开销。为适配不同计算任务的特性,设计了一种融合数据并行和模型并行的混合并行策略,将数据与模型的并行计算能力有效结合,使得不同子模块可以协同高效运行,既满足训练过程中的任务并行需求,又能降低通信开销,提高整体训练效率。



$F_{0,0}$、$F_{0,1}$、$F_{1,0}$、$F_{1,1}$—微批次粒度的层前向数据；$B_{0,0}$、$B_{0,1}$、$B_{1,0}$、$B_{1,1}$—微批次粒度的层反向数据；

a_i、b_i、c_i、d_i（$i=1,2,3$）—梯度参数。

图 1　面向多模态模型的分区混合并行优化方法系统架构

Fig. 1　System architecture of the partitioned hybrid parallelization method for multimodal models

在计算任务调度资源分配方面，提出了一种基于动态规划计算任务调度优化算法，（dynamic programming-based task scheduling optimization algorithm，DP-TSA），以实现计算资源的动态分配与高效利用。该算法通过分析各任务子模块的计算需求、资源占用情况和任务依赖关系，动态地将计算资源分配给不同并行任务。通过动态规划的方式，系统能够在异构资源环境下优化任务调度，平衡各计算节点的运行负载，避免资源的闲置与过载，提高整体系统的资源利用率与计算性能。

2　模块分区

通过模块分区，实现多模态模型子模块划分，为具有异构特性的不同子模块选择适合的并行方式奠定基础。

2.1　子模块异构性特点

多模态模型的子模块在参数规模、输入数据类型和计算复杂性上存在显著异构性。

（1）参数规模方面。以对比语言-图像预训练模型（CLIP）为例[24]，视觉编码器使用视觉 Transformer 基础版 32（ViT-B32）配置时，参数规模占总参数量的 58.1%；升级至大型（ViT-L14）配置后，占比升至 82.7%，参数总量达 3.07×10^8。这种参数占比的偏置，会加剧计算量与显存占用的偏置。在训练过程中，这种不对称会被放大，使负载更大或更受限的一侧成为瓶颈，从而拉低整体资源利用率。

（2）输入数据方面。图像输入（如 512×512 像素）对计算能力和显存要求高，而文本输入（如 77 个单词）计算需求小，但需快速处理大批次数据。这种差异加剧了子模块间的异构性，导致计算负载分布不均。

（3）计算复杂性方面。视觉编码器任务（如特征提取、分辨率调整）计算复杂度高，需复杂卷积或自注意力计算；文本编码器任务（如嵌入、序列建模）计算复杂度低，但对低延迟和高吞吐性能要求高。这种复杂度差异影响计算资源需求，可能引发额外通信开销。

以上子模块在参数规模、输入数据特性和任务复杂度上的异构性使多模态模型在训练中面临负载不均衡、通信开销过高、并行方式与计算特性不适应、资源利用率低和性能不高等挑战。

2.2　模块分区方法

针对子模块异构性的问题，合理进行模块分区是优化多模态模型训练性能的关键。模块分区核心目标是通过对深度神经网络（DNN）子模块的划分与调度，提升其独立性和负载均衡能力，从而优化资源利用效率、降低通信开销，实现高效的分布式训练。本文模块分区的整体框架如图 2 所示。

首先，对多模态深度神经网络模型的整体架构进行深入分析，明确各子模块的具体功能和相互依赖关系，明确子模块通过内部算法和网络层处理各自的数据类型，并在某些情况下需要进行数据交换或协同工作。在这一过程中，参数规模和计算需求被视为影响显存占用和计算复杂度的核心因素。其次，通过深入的架构剖析，识别出各子模块的异构特性，例如视觉编码器通常表现为高计算复杂度和高

参数规模，而文本编码器的计算复杂度和数据需求相对较低。

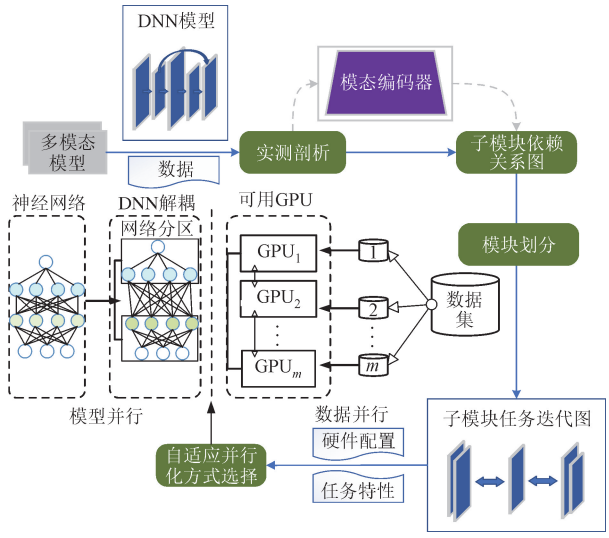


图 2 模块分区框架

Fig. 2 Module partitioning framework

2.2.1 模块依赖图构建及模块划分

基于异构特性分析结果，构建模块依赖关系图，表明子模块间的通信需求、数据流动路径及计算负载分布，有效识别可能的通信瓶颈和负载集中区域。图 3 展示了多模态模型中文本和图像数据的子模块依赖划分，表示了模型内部结构、依赖关系及数据处理流程。

在该模型中，文本数据通过文本编码器提取出代表文本语义的特征向量 $[T_1, T_2, \cdots, T_N]$；图像数据则通过图像编码器，生成具有区分性的视觉特征向量 $[I_1, I_2, \cdots, I_N]$。完成特征提取后，文本与图像特征通过跨模态交互模块进行对比融合，实现模态间的信息互补与增强，最终输出模型结果。

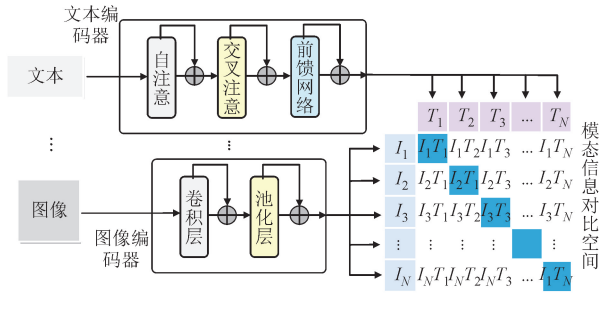


图 3 子模块依赖图

Fig. 3 Submodule dependency graph

为了量化子模块间的通信需求和计算负载，将每个子模块 i 作为图中的一个节点，标注其计算资源消耗 $\boldsymbol{W}(i)$，公式如下

$$\boldsymbol{W}(i) = [\, C(i), P(i)] \tag{1}$$

式中：$C(i)$ 表示需要执行的计算量，i 为子模块；$P(i)$ 为参数规模。

子模块之间的依赖关系通过有向边 E 表示，边上的权重 F_{ij} 表示模块间的数据传输效率，具体表达式如下

$$F_{ij} = \frac{D_{ij}}{T_{ij}} \tag{2}$$

式中：D_{ij} 为子模块 i 向目标子模块 j 传输的数据量；T_{ij} 为传输时间间隔。

通过分析图中数据参数和计算负载集中的节点，可以识别潜在的通信瓶颈和性能瓶颈。所有信息均记录在字典键值中，为后续分析提供数据支持。

基于收集的信息流和收集的子模块状态信息，深入分析系统内部关系，旨在为模型切分优化提供支持，利用动态规划算法和图论中最小割集理论找到最佳分割点。最小割集理论用于识别图中连接最紧密的节点集合，这些集合的分割可以最大程度地减少边的权重总和，即降低通信开销，可表示为

$$\min \sum_{(i,\, j) \in E} F_{ij} \tag{3}$$

根据最优分割点数组，模型划分为多个独立的执行阶段，确保每个阶段能够高效运行。在切分过程中，边界条件的设计遵循子模块最小独立性、通信开销合理化和资源分配合理性 3 项核心原则。具体而言，每个子模块需具备完整的计算逻辑和数据流闭环能力，并且输入输出维度与原始模型严格一致，可表示为

$$\begin{cases} I(G_m) = I_{\text{orig}} \\ O(G_m) = O_{\text{orig}} \end{cases} \tag{4}$$

式中：G_m 表示划分了第 m 个后的独立子模块，由原始模型中的一组子模块节点组成；I_{orig} 和 O_{orig} 为原始输入输出维度。该约束保证了子模块可以脱离其他模块独立运行，避免跨模块间因依赖关系复杂而引发执行顺序冲突，同时保障数据流的连续性和完整性。

此外通过优化分区粒度，将跨模块通信开销控制在系统可接受范围内，防止因过多的跨模块通信而导致性能下降，可表示为

$$C_{\text{comm}} \leqslant C_{\max} \tag{5}$$

式中：C_{comm} 为当前跨模块通信量；$C_{\max}$ 是系统所能容忍的最大通信开销上限。

各划分后的独立子模块分区 G_m 的资源需求也必须满足系统硬件资源的可用性限制包括计算单元、内存带宽等，以避免因资源不足导致性能瓶颈，可表示为

$$R(G_m) = f\Big(\sum_{v_i \in G_m} C(v_i),\ \sum_{v_i \in G_m} P(v_i)\Big) \tag{6}$$

式中：$R(G_m)$ 为划分后模块所需资源；函数 $f(\cdot)$ 综合评估了 G_m 内所有子模块的计算量和参数规模，以确定整个子模块的资源需求。

针对参数规模较小且计算需求不高的辅助模块，不再进行单独分区，采取与主模块合并的策略，既消除了冗余的跨设备通信开销，又避免了因微小分区导致的资源碎片化问题。通过这种切分方法，在保持系统原始依赖关系的同时，实现了计算负载的均衡分布，最终形成的多个独立子模型既具备处理特定数据类型的专用性，又确保了整体系统的高效协同运行。

2.2.2　*并行方式确定*

在完成模块划分后，根据各子模块任务迭代关系，结合硬件资源的分布情况，设计自适应并行化方式选择机制，以针对划分后的子模块进行定制化的并行处理。

对于参数规模大、计算需求高的子模块（如视觉编码器），采用模型并行策略。通过将这些子模块的负载分散到多个设备上，以支持大规模模型的扩展训练，有效避免单个设备资源瓶颈的问题。例如，图像编码器由于其高计算复杂度和较大的参数规模，会被分割成多个部分分布在不同的设备上，在确保高效处理的同时减少单一设备的负担。

对于需要处理大批量数据或数据流量较高的子模块（如文本编码器），则选择数据并行策略。通过在多个设备上并行处理大量小批次的数据，来提高吞吐量，同时通过优化模块间的部署降低跨设备通信延迟。

3　混合并行优化

完成模块分区和并行方式选择后，通过引入流水线技术和微批量处理实现模型并行中的通信与计算重叠；数据并行则采用 Ring Allreduce 的多 GPU 同步随机梯度下降（SGD）进行高效梯度聚合，保障训练的一致性与稳定性。采用模型并行与数据并行相结合的混合优化策略，增强了多模态模型训练的灵活性和适应性，以充分利用计算资源，进一步加快训练速度。

3.1　模型流水线并行优化

在处理计算负载分散的多模态大规模神经网络模型训练时，传统单一多加速器顺序执行方式会导致不同加速器间出现空闲状态，产生“气泡”开销，延长训练时间。本文流水线并行优化方法，通过将数据批量划分为更小的微批量，让多个 GPU 并行处理，实现密集迭代训练，从而提升 GPU 利用率，减少气泡开销，降低模型并行训练的时间开销。具体优化方法如下所述。

（1）模型层划分及资源分配。根据各层的计算复杂度和通信需求，将模型层独立划分并分配到最适合的设备上。高计算复杂度的层（如 CNN 中的卷积层或 Transformer 中的自注意力层）被单独划分为独立分区，部署于高性能 GPU 上，以充分利用其计算能力；而低计算复杂度的层（如激活函数层）则被分配至资源较少的设备上，以实现负载均衡。通过合理划分模型层并分配资源，可以有效减少设备间的通信开销，提高设备利用率，从而提高整体训练效率。

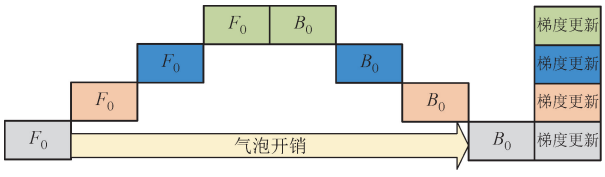
（2）任务切分与配置管理。将各个子网络层依次放置在不同的 GPU 上，并通过引入总体模型任务切分迭代规范配置文件，确保子网络层之间的数据传输和计算顺序与原始模型一致。通过任务切分和配置管理，保障模型计算逻辑正确性，避免因设备间的通信延迟而导致的性能下降。

（3）微批量处理与 CUDA 流优化。将原始小批量数据切分为多个更细粒度的微批量，并依次分配到已部署子模块网络层 S_i 的对应 GPU 设备 GPU_i 上进行密集迭代执行。通过非默认 CUDA 流设置，为每个微批量的输入数据 i_{S_i} 和输出数据 o_{S_i} 分配独立的计算流，并明确任务间的依赖关系，从而实现后续 GPU 计算与前序通信操作的重叠，避免因等待数据输入而导致的空闲时间。此外，在每次完整的小批量迭代结束后，统一同步更新模型参数，确保所有 GPU 上的模型状态一致，以加速训练收敛。该方法有效减少了设备“气泡”开销，提升了 GPU 利用率和整体训练效率。算法的时间复杂度为 $O(LN_{\mathrm{G}}+B_{\mathrm{s}}+ML+P)$，其中 L 是模型层数，N_{G} 是 GPU 数量，B_{s} 是批量大小，M 是微批量大小，P 是参数总量。

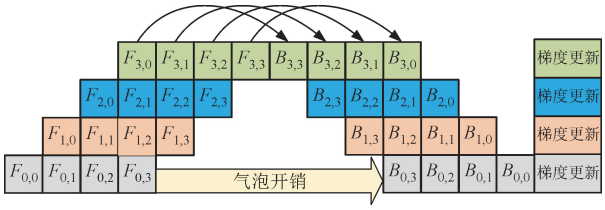
图 4 直观地对比了传统模型并行与流水线并行在计算效率上的显著差异。在图 4(a)中，传统模型并行采用串行处理方式，整个批次数据以完整计算单元的形式在设备间顺序传递，如从完整前向阶段 F_0 到完整反向阶段 B_0。每个设备必须等待前一阶段完全结束后才能开始计算，导致设备闲置时间过长。图中的灰色区域代表设备闲置的气泡开销，尤其在 F_0 到 B_0 的过渡阶段，设备间传递的巨大灰色箭头生动体现了梯度更新前的漫长等待期，使得设备利用率长期处于较低水平。

相比之下，图 4(b)展示了流水线并行通过微批次拆分实现的优化。完整批次被分解为多个微批次单元（如 $F_{0.0}$ 至 $F_{0.3}$）。这些微批次在设备间形成重

叠处理的流水线。具体来说，当设备 1 正在处理第 n 个微批次的前向计算时，设备 2 可以同步执行第 $n-1$ 个微批次的反向计算（如 $B_{0,0}$）。图中的微型箭头链表明，气泡开销被压缩至仅剩微批次间的间隙时间。这种优化显著提高了梯度更新的频率，使得设备利用率大幅度提升。



(a)传统模型并行气泡开销



(b)流水线并行气泡开销

图 4 传统模型并行和流水线并行气泡开销的对比

Fig. 4 Comparison of bubble overhead between traditional model parallelism and pipeline parallelism

总结来说，传统模型并行由于串行处理导致明显的计算间隙（气泡开销），而流水线并行通过微批次拆分和重叠计算，形成连续的计算流，大幅减少设备空闲时间，使梯度更新更频繁，从而显著增强训练效率。

3.2 数据并行同步随机梯度下降优化

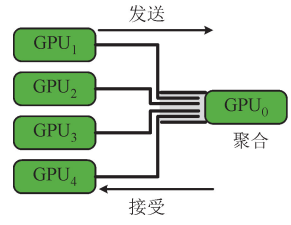
在采用数据并行对数据密集型模型进行并行训练时，训练数据被划分为多个小批量，分配到不同的 GPU 节点上，每个 GPU 节点独立完成前向传播和反向传播计算。为确保全局模型参数的一致性更新，采用同步 SGD 算法，并通过 Ring Allreduce 机制进行全局梯度聚合和参数更新。

同步 SGD 是一种分布式深度学习优化算法，其核心在于通过严格的全局同步机制，确保所有计算节点使用一致的模型参数进行更新，从而保障训练过程的收敛稳定性。

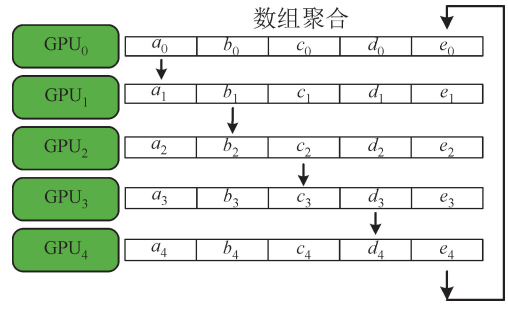
具体而言，训练数据集 D 被均匀划分为 P 个子集 $\{D_1, D_2, \cdots, D_P\}$，分别分配给 P 个 GPU 节点。每个节点从其对应的子集 D_P 中加载小批量数据 B，并独立完成前向传播和反向传播计算，得到局部梯度 g_p。完成模型的计算任务后，进行全局模型参数更新，各节点将本地计算出的局部梯度进行高效聚合，生成统一的全局梯度 G_{sync}。本文采用 Ring Allreduce 作为梯度聚合通信机制，相比传统的树形聚合或集中式聚合方式，该方法具有更高的效率和良好的可扩展性。其通信复杂度为 $O(2N/P)$。这种低复杂度使它能够高效利用通信带宽降低通信开销。同时，环形拓扑结构设计避免了传统集中式方法存在的中心节点瓶颈问题，提升了系统的扩展性和鲁棒性。

Ring Allreduce 包含 Scatter-Reduce（梯度聚合）和 All-Gather（结果广播）两个阶段，确保所有节点最终都能获得一致的全局梯度 G_{sync}。随后，各节点基于 G_{sync} 对本地模型参数 W 进行同步更新。整个训练过程循环执行从数据分配到参数更新的完整流程，直至模型达到预定的收敛标准。整体时间复杂度为：$O(B_s/PL+2N/P+P)$，B_s 是批量大小，L 是模型层数，N 是模型参数总量，P 是 GPU 数量。

图 5 给出了传统参数聚合方法与 Ring Allreduce 全局梯度聚合的对比。图 5(a)中所有 GPU 的数据通过 Send 操作发送到一个中心节点（GPU_0），在该中心节点进行 Reduce 操作合并数据，然后将结果分发回各个 GPU。这种中心化聚合方式在大规模系统中容易使中心节点成为瓶颈，导致显著的通信开销。图 5(b)展示了 Ring Allreduce 聚合方式，所有 GPU 形成一个逻辑环形结构，每个 GPU 仅与其相邻的两个 GPU 通信，梯度数据$[a_i, b_i, c_i, d_i, e_i]$在环中逐步传递、计算，最终得到完整的全局聚合结果。这种方法避免了中心节点瓶颈问题，提高了通信效率和系统的扩展性。



(a)传统通信聚合



(b)Ring Allreduce 通信聚合

图 5 传统通信聚合与 Ring Allreduce 通信聚合的对比

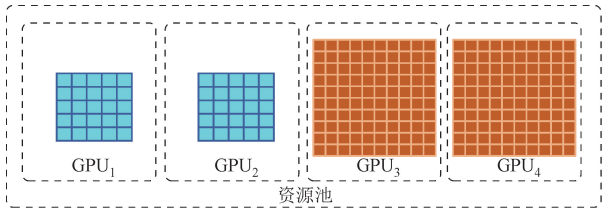
Fig. 5 Comparison of traditional communication aggregation and Ring Allreduce communication aggregation

4　动态任务调度优化

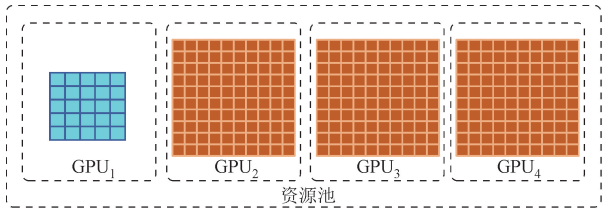
在多模态深度学习任务中，随着模型规模和计算需求的增长，不同任务子模块间的计算负载差异显著增加。传统的静态资源分配策略采用统一标准将资源均分给所有子模块任务，忽略了实际计算需求差异，导致资源利用率低、训练时间长。为解决这一问题，提出了一种基于动态规划任务调度优化算法 DP-TSA。根据各子模块任务的实际需求合理映射资源，以提升系统的整体性能和资源利用效率。

4.1　动态资源分配分析

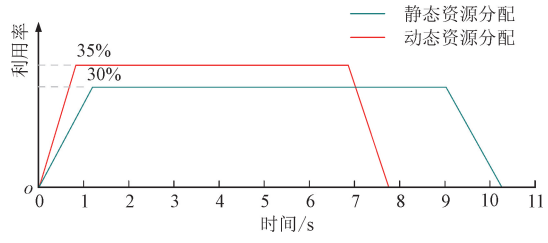
为了分析动态分配资源的效果，首先构建了如图 6 所示的基于假设性能指标的示例图。图 6(a)为静态资源分配的结果示意图，可见：由于未能考虑任务子模块的具体需求，较小子模块尽管其参数量少、计算任务轻、所需资源有限；在静态分配中，被赋予与较大子模块相同的资源量，导致资源过剩，其 GPU 利用率仅为 25%，而较大子模块的利用率为 35%，整体 GPU 平均利用率为 30%。这种资源分配与实际需求的不匹配，使得较小子模块成为性能瓶颈，导致整体资源利用率低下、训练时间延长，从而影响了整体训练效率。



(a)静态资源分配



(b)动态资源分配



(c)资源分配效果对比

图 6　静态资源和动态资源分配下的调度效果对比

Fig. 6　Comparison of scheduling effects under static and dynamic resource allocation

图 6(b)为动态资源分配策略的优化效果，可见：该策略通过灵活调整资源分配比例，根据实际需求将计算资源精准映射到各个子模块；较小子模块避免了过度分配，而较大子模块通过自适应分配满足了其较高的计算需求，实现了更精细的资源管理；动态调整使较小子模块的 GPU 利用率从 25%提升至 29%，同时确保较大子模块不受资源不足的限制，提高了整体资源利用率和训练效率。

此外，动态资源分配通过最小化模块间训练时间的方差，实现了更均衡的训练负载分布，减少了模块间的训练时间差异，避免了训练较快的模块因等待较慢模块而产生的延迟，使各子模块的训练时间更加一致。

4.2　任务调度数学模型

本文针对多模态模型训练中的任务调度与资源管理问题，构建了如下的数学优化模型。

设系统包含 M 个子模块和 N 张 GPU，每个子模块 i 的计算需求紧密依赖于其分配的 GPU 数量 x_i。定义二元变量 y_{ij} 表示为第 j 张 GPU 是否分配给第 i 个子模块的变量，其表达式为

$$y_{ij}=\begin{cases}1, & \text{如果第 } j \text{ 张 GPU 分配给第 } i \text{ 个子模块}\\ 0, & \text{否则}\end{cases} \tag{7}$$

式中：如果 GPU_j 被分配给第 i 个子模块，则 $y_{ij}=1$；否则 $y_{ij}=0$。子模块 i 的总 GPU 分配数量 x_i 表示为

$$x_i=\sum_{j=1}^{N} y_{ij},\ \forall i\in\{1,2,\cdots,M\} \tag{8}$$

本文以最小化所有子模块训练时间 T 的方差为目标，实现训练时间 T 的均衡化，具体目标函数定义为

$$\min \operatorname{var}(T)=\frac{1}{M}\sum_{i=1}^{M}(t_i(x_i)-\overline{T})^2 \tag{9}$$

$$\text{s.t.}\begin{cases}\mathrm{C1}:\ x_i=\sum\limits_{j=1}^{N} y_{ij},\ \forall i\in\{1,2,\cdots,M\}\\ \mathrm{C2}:\ \sum\limits_{i=1}^{N} y_{ij}\leqslant 1,\ \forall j\in\{1,2,\cdots,N\}\\ \mathrm{C3}:\ \sum\limits_{i=1}^{M} x_i=N\\ \mathrm{C4}:\ x_i\geqslant 1,\ x_i\in Z\end{cases} \tag{10}$$

式中：$t_i(x_i)$ 表示子模块的训练时间是分配 GPU 数量 x_i 的非增函数，即增加 GPU 分配通常会缩短训练时间，但边际收益递减；$\overline{T}$ 为所有子模块训练时间

的均值;约束条件 C1 为每个子模块的 GPU 分配数量为其所需总和;约束条件 C2 为每张 GPU 只能分配给一个子模块;约束条件 C3 为所有子模块分配的 GPU 总数等于系统 GPU 总数;约束条件 C4 为每个子模块至少分配 1 张 GPU 且分配数量为整数。

基于上述数学模型,设计了基于动态规划计算任务调度优化算法。该算法将资源调度问题抽象为一系列阶段性决策问题,通过递归求解每个阶段的最优解,逐步构建出全局最优方法。本方法确保了在满足各种约束的前提下最大化资源利用效率,有效实现计算负载的均衡分布。

算法优化执行过程如下。

(1)定义状态转移方程。设 i 为子模块编号,k 为分配给这些子模块的 GPU 总数量。定义状态函数 $f[i][k]$ 表示前 i 个子模块在分配 k 张 GPU 时的最小训练时间方差,$t[i][k]$ 用于记录对应的状态转移路径,以便最终回溯并确定每个子模块的 GPU 分配数量。初始时,将 $f[i][k]$ 的所有元素设为无穷大,表示初始状态下的最小方差尚未确定;将 $t[i][k]$ 的所有元素设为零,表示初始状态下无路径记录。$f[i][k]$ 状态转移方程为 $f[i][k]=\min\limits_{x_i\leqslant k}\{f[i-1][k-x_i]+\mathrm{var}(T_i)\}$,它表示在为第 i 个子模块分配 x_i 张 GPU 时,通过考虑上一阶段的最优状态递推当前状态的最优解,确保每个子模块获得最适合的 GPU 资源分配。算法初始状态设定 $f[0][0]=0$,即没有子模块且没有 GPU 时的训练时间方差为 0,并且 $f[0][k]=\infty$, $\forall k\in\{1,2,\cdots,N\}$,表示没有子模块但有 GPU 时的训练时间方差视为无穷大。

(2)计算最优解。使用动态规划递推计算所有状态 $f(i,k)$。对每个子模块 i 和可能的 GPU 分配数量 k,尝试分配 x_i 张 GPU,计算训练时间方差。若 $f[i-1][k-x_i]+\mathrm{var}(T_i)$ 小于当前 $f[i][k]$,则更新 $f[i][k]$ 和 $t[i][k]$ 数组为 x_i。最终得到全局最优解 $f[M][N]$。

(3)任务调度优化与映射。初始化剩余 GPU 数量 k 为总 GPU 数 N,创建数组 Z^* 存储分配结果。依据路径表 $t[i][k]$ 从最后一个子模块向前回溯,若 x_i 不为 -1,则将 x_i 存入 Z^* 并更新剩余 GPU 数量 k_1。回溯完成后,Z^* 中的分配方案即为最优解,实现资源的合理调度,提升多模态 DNN 训练效率。

算法的总时间复杂度为:$O(MN^2)$,其中 M 是子模块的数量,N 是可用 GPU 的数量。这一复杂度主要由递推计算阶段决定,因为每个状态 $f[i][k]$ 需要计算 $O(N)$ 个子问题,而状态总数为 $O(MN)$。

任务调度优化具体实现如下。

算法 1 基于动态规划的任务调度优化算法

输入:子模块数量 M,可用 GPU 数 N,训练时间 $T[i][x]$

输出:最优资源分配方案 $Z^*=\{x_1,x_2,\cdots,x_N\}$

1. 初始化状态表 $f[i][k]$ 和路径表 $t[i][k]$
2. $f[0][0]\leftarrow 0$,$f[0][k]\leftarrow\infty$,$\forall\ k\in\{1,2,\cdots,N\}$
3. $f[i][k]\leftarrow\infty$
4. for i from 1 to M do
5. for k from 0 to N do
6. for x from 0 to k do
7. If $f[i-1][k-x]+T[i][x]<f[i][k]$ then
8. $f[i][k]\leftarrow f[i-1][k-x]+T[i][x]$
9. $t[i][k]\leftarrow x$
10. end if
11. end for
12. end for
13. end for
14. $k\leftarrow N$,$Z^*\leftarrow[0,0,\cdots,0]$
15. for i from M to 1 do
16. if $t[i][k]\neq -1$ then
17. $x_i\leftarrow t[i][k]$
18. $Z^*[i-1]\leftarrow x_i$
19. $k_1\leftarrow k-x_i$
20. end if
21. end for
22. return Z^*

5 实验验证

5.1 实验设置

(1)实验环境:实验在天津超算的 HPC4-GPU 分区进行,硬件环境为 1 个配备 8 卡 NVIDIA HGX A100 GPU 的节点,节点内的 GPU 通过 NVLink 互联。软件运行环境为 CUDA 12.1 和 PyTorch 2.4.0。

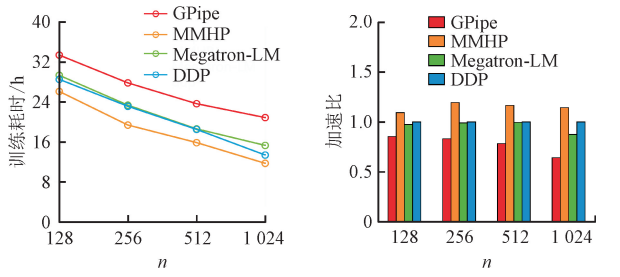
(2)数据集和模型:实验基于 PyTorch 框架,使用 MSCOCO 数据集,该数据集包含 60 万对图像-文本样本,广泛用于图像-文本匹配和检索任务。实验

中，图像输入被统一调整为 224×224 像素，文本长度限制为 77 个文本单元。模型选用 CLIP[24] 和引导式语言-图像预训练模型（BLIP）[25] 两个主流多模态预训练模型作为基线模型，其中 CLIP 的视觉编码器采用残差网络-101（ResNet-101）架构，BLIP 的视觉编码器采用 ViT-B32 架构，文本编码器则统一采用 Transformer 架构。实验旨在评估本文方法在多模态任务中的训练效率、性能表现及对不同神经网络架构的适配能力。

5.2　实验结果分析

5.2.1　训练耗时对比

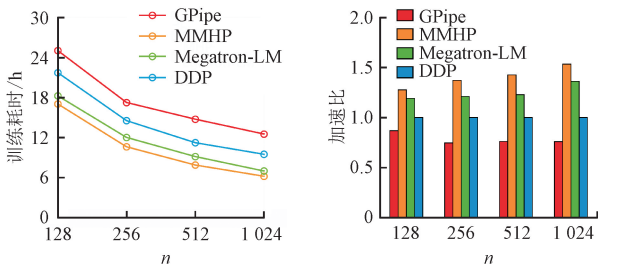
为验证本文方法（MMHP）在提升多模态模型并行训练效率方面的有效性，将其与 3 种主流并行训练方法分布式数据并行（DDP）、张量并行（Megatron-LM）以及流水线并行（GPipe）进行了对比实验。实验在 CLIP 和 BLIP 两个经典多模态预训练模型开展，每种方法均训练 70 个轮次，对比其训练耗时与加速比，其中 n 为批量大小。实验结果如图 7、图 8 所示。



（a）在 CLIP 模型上训练耗时　（b）在 CLIP 模型上训练加速比

图 7　MMHP 与主流并行方法在 CLIP 上训练耗时和加速比对比

Fig. 7　Training time comparison between MMHP and mainstream parallel methods on CLIP model



（a）在 BLIP 模型上训练耗时　（b）在 BLIP 模型上训练加速比

图 8　MMHP 与主流并行方法在 BLIP 上训练耗时和加速比对比

Fig. 8　Training time comparison between MMHP and mainstream parallel methods on BLIP model

实验结果表明，本文方法在训练耗时和加速性能上均优于其他方法。在 CLIP 模型中，其视觉编码器采用 ResNet-101 架构，计算密集且网络层次较深，其他并行方法易受负载不均和通信开销影响，MMHP 通过模块化划分和动态资源调度，有效缓解了计算瓶颈，减少了 GPU 空闲时间，使训练耗时降低 30%以上，同时保持较高加速比，展现出良好的扩展性和适配能力。在 BLIP 模型中，其视觉编码器采用 ViT-B/32，具有更好的并行性和参数分布均衡性，更适合分布式并行优化。MMHP 通过高效的混合并行策略，进一步优化了跨模态交互中的通信效率和资源调度，尤其在较大 batch size 条件下，充分发挥了其模块划分与调度机制的优势，提升了训练速度和可扩展性。此外，与其他方法相比，GPipe 虽在大规模模型训练中表现出一定的稳定性，但其固定的阶段划分机制及较弱的异构设备调度能力，在多模态任务中限制了其训练效率和资源利用率。例如，在 BLIP 模型上，当批量大小为 1 024 时，其训练时间是 MMHP 方法的两倍。

总体来看，MMHP 在 CLIP 和 BLIP 模型上的表现均优于 DDP、Megatron-LM 和 GPipe，展现出更强的适应性与性能优势，相较于 GPipe 训练速度最大可提升 2 倍。

5.2.2　训练吞吐量对比

为验证本文方法在提升训练吞吐量方面的有效性，本文将其与 DDP、Megatron-LM 和 GPipe 进行了对比实验。实验在 CLIP 和 BLIP 模型上进行，训练 70 个轮次，并在不同批量大小下评估性能。

表 1 实验结果表明，本文方法 MMHP 在 CLIP 和 BLIP 模型的训练吞吐量上均高于其他方法。对于 CLIP 模型，其计算复杂性更高且通信开销较大，其他并行化方法提升效果有限，但 MMHP 通过优化模块分区和资源调度，缓解了负载不均和通信瓶颈，在批量大小为 1 024 时，系统吞吐量达到每秒 998.23 个样本。对于 BLIP 模型其参数均衡性和并行性更优，结合 MMHP 高效的调度策略与通信优化，在批量大小为 1 024 时，吞吐量进一步提升至 1 727.30。

总体而言，本文方法通过精细的模块分区与资源调度优化，提升了多模态模型的并行效率和训练吞吐量。不论是模型结构较重的 CLIP，还是更适合并行训练的 BLIP 中均表现出色。

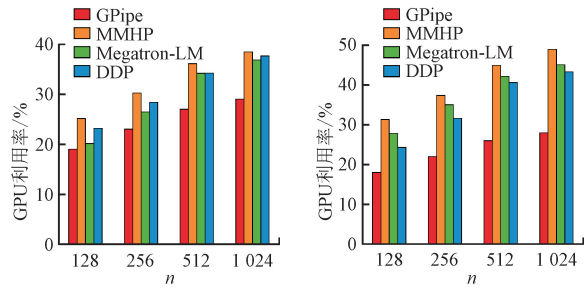
表 1 不同并行化方法在 CLIP 和 BLIP 上的平均吞吐量

Table 1 Average training throughput of CLIP and BLIP models under different parallel strategies

基准模型	批量大小	样本处理速率/s^{-1}			
		DDP	Megatron-LM	MMHP	GPipe
CLIP	128	409.56	398.37	447.47	341.27
	256	505.24	500.22	603.03	430.15
	512	631.59	627.93	736.14	534.84
	1 024	873.86	763.62	998.23	683.07
BLIP	128	537.86	640.22	686.01	422.31
	256	804.14	972.87	1 101.80	607.53
	512	1 041.73	1 278.55	1 484.20	729.12
	1 024	1 232.70	1 677.73	1 727.30	861.59

5.2.3 GPU 利用率对比

为验证本文方法在提升 GPU 利用率方面的有效性，将其与 DDP、Megatron-LM 和 GPipe 在 CLIP 和 BLIP 模型上进行了对比实验。所有方法均在不同批量大小下训练 70 个轮次，结果如图 9 所示。



(a)在 CLIP 模型上的利用率 (b)在 BLIP 模型上的利用率

图 9 MMHP 与主流并行方法在 CLIP 和 BLIP 模型上的 GPU 利用率对比

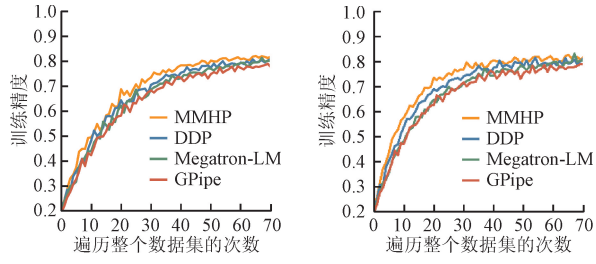
Fig. 9 Comparison of GPU utilization between MMHP and mainstream parallel methods on CLIP and BLIP models

从实验结果可以看出，MMHP 方法在不同批量大小设置下，GPU 利用率总体优于其他方法。在 CLIP 模型中，GPU 利用率随着批量大小的增加逐渐提升，在批量大小为 128 时，MMHP 的 GPU 利用率为 25.13%，分别高于 Megatron-LM 的 20.12%和 DDP 的 23.17%以及 GPipe 的 19.1%；随着 batch size 增大至 1 024，MMHP 的 GPU 利用率进一步提升至 38.49%。对于 BLIP 模型，整体 GPU 利用率更高，主要得益于其视觉 Transformer 基础版 32 配置架构具备更高的计算密度和良好的硬件适配性。在批量大小为 128 时，MMHP 的 GPU 利用率为 31.33%，当批量大小增至 1 024 时，MMHP 的 GPU 利用率达到 48.94%，分别比 Megatron-LM、DDP 和 GPipe 高出 8.5%、13.01%和 20.94%。

MMHP 通过精细化的模块划分与动态资源调度策略，能够根据不同模型结构灵活调整并行化方式，实现更高效的通信与负载均衡，减少了任务等待时间和资源空闲，提升了系统的稳定性与适应能力，从而在多种训练场景下均展现出优异的 GPU 利用表现。

5.2.4 训练精度对比

为验证本文方法在模型训练精度方面的有效性，本文将其与 DDP、Megatron-LM 和 GPipe 3 种主流并行训练方法进行了对比实验。实验在 CLIP 和 BLIP 两种多模态深度神经网络模型上进行，每种方法均训练 70 个训练轮次，以比较其在训练精度方面的表现。实验结果如图 10 所示。



(a)在 CLIP 模型上训练精度 (b)在 BLIP 模型上训练精度

图 10 不同并行化方法在 CLIP 和 BLIP 上的训练精度对比

Fig. 10 Comparative training accuracy of CLIP and BLIP models across different parallel strategies

从图 10 中可以看出采用 MMHP 方法不仅能够减少训练时间，同时保持了与其他并行化策略相当甚至更高的训练精度。在 CLIP 模型上训练过程达到稳定后，MMHP 实现了高达 82.3%的最大训

练精度。这表明，尽管 MMHP 在计算资源分配和任务调度上进行了优化调整，但并未牺牲模型的学习能力或泛化性能。在 BLIP 上的训练中，MMHP 同样展现了在提升训练效率的同时，确保了模型的高精度输出。特别是在处理复杂的跨模态交互任务时，MMHP 通过精细的模块划分与资源调度优化，有效减少了通信开销，缓解了负载不均衡问题，从而增强了模型的稳定性和准确性。

综合来看，MMHP 在保证训练效率提升的前提下，并未以牺牲模型精度为代价，展现出良好的精度-效率平衡能力，适用于对性能与精度均有较高要求的多模态训练任务。

6　结　论

本文提出了一种面向多模态深度神经网络训练的分区混合并行方法(MMHP)，旨在优化传统并行化方法在多模态深度神经网络训练中存在的负载不均、通信开销大及资源利用率不足等问题。该方法通过深入分析多模态模型中各子模块的异构特性，构建模块依赖关系图，识别并优化关键切分点，有效缓解了计算负载集中和通信瓶颈问题。在此基础上，MMHP 依据子模块的参数规模、计算需求和任务特性，定制数据并行与模型并行策略，实现对异构任务的高效适配与模块级并行执行。此外，设计了一种基于动态规划的资源调度优化算法，动态分配计算资源，实现计算任务与资源的合理映射，从而提升资源利用效率，减少资源浪费和等待时间。实验结果表明，与 3 种主流并行训练方法(DDP、Megatron-LM 和 GPipe)相比，本文所提方法在不影响模型训练精度的情况下，从训练时间、吞吐量和 GPU 利用率等方面均优于以上方法，缓解了传统方法所面临的内存限制、通信复杂度高、适应性不足问题，展现出良好的扩展性与系统适配能力。

参考文献：

[1] WANG Jiaqi, JIANG Hanqi, LIU Yiheng, et al. A comprehensive review of multimodal large language models: performance and challenges across different tasks [EB/OL]. (2024-08-02)[2024-12-18]. https://arxiv.org/abs/2408.01319.

[2] LIANG Zijing, XU Yanjie, HONG Yifan, et al. A survey of multimodel large language models [C]//Proceedings of the 3rd International Conference on Computer. New York, NY, USA: ACM, 2024: 405-409.

[3] LI Jian, LU Weiheng, FEI Hao, et al. A survey on benchmarks of multimodal large language models[EB/OL]. (2024-09-06)[2024-12-19]. https://arxiv.org/abs/2408.08632.

[4] ZHANG Min, LI Juntao. A commentary of GPT-3 in-MIT Technology Review 2021 [J]. Fundamental Research, 2021, 1(6): 831-833.

[5] TOUVRON H, LAVRIL T, IZACARD G, et al. LLaMA: open and efficient foundation language models [EB/OL]. (2023-02-27)[2024-12-19]. https://arxiv.org/abs/2302.13971.

[6] HUANG Jun, ZHANG Zhen, ZHENG Shuai, et al. DISTMM: accelerating distributed multimodal model training [C]//Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation. USA: USENIX Association, 2024: 1157-1171.

[7] 王恩东，闫瑞栋，郭振华，等. 分布式训练系统及其优化算法综述 [J]. 计算机学报，2024，47(1)：1-28.
WANG Endong, YAN Ruidong, GUO Zhenhua, et al. A survey of distributed training system and its optimization algorithms [J]. Chinese Journal of Computers, 2024, 47(1): 1-28.

[8] 卢凯，赖志权，李笙维，等. 并行智能训练技术：挑战与发展 [J]. 中国科学(信息科学)，2023，53(8)：1441-1468.
LU Kai, LAI Zhiquan, LI Shengwei, et al. Parallel intelligent computing: development and challenges [J]. Scientia Sinica (Informationis), 2023, 53(8): 1441-1468.

[9] 王帅，李丹. 分布式机器学习系统网络性能优化研究进展 [J]. 计算机学报，2022，45(7)：1384-1411.
WANG Shuai, LI Dan. Research progress on network performance optimization of distributed machine learning system [J]. Chinese Journal of Computers, 2022, 45(7): 1384-1411.

[10] LI Dongsheng, LI Shengwei, LAI Zhiquan, et al. A memory-efficient hybrid parallel framework for deep neural network training [J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 35(4): 577-591.

[11] XU Lang, ANTHONY Q, ZHOU Qinghua, et al. Accelerating large language model training with hybrid GPU-based compression [C]//2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid). Piscataway, NJ, USA: IEEE, 2024: 196-205.

[12] BIAN Song, LI Dacheng, WANG Hongyi, et al. Does compressing activations help model parallel training? [J]. Proceedings of Machine Learning and Systems, 2024,

6：239-252.

[13] CHEN Y C，LI Linjie，YU Licheng，et al. UNITER：UNiversal image-TExt representation learning [C]// Computer Vision-ECCV 2020. Cham，Switzerland：Springer International Publishing，2020：104-120.

[14] XUE Zhenliang，HU Hanpeng，CHEN Xing，et al. PipeWeaver：addressing data dynamicity in large multimodal model training with dynamic interleaved pipeline [EB/OL]. [2025-04-05]. https://arxiv. org/abs/2504. 14145.

[15] HUANG Yanping，CHENG Youlong，BAPNA A，et al. GPipe：efficient training of giant neural networks using pipeline parallelism [C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems. Red Hook，NY，USA：Curran Associates Inc.，2019：103-112.

[16] ALAYRAC J B，DONAHUE J，LUC P，et al. Flamingo：a visual language model for few-shot learning [C]//Advances in Neural Information Processing Systems. Red Hook，NY，USA：Curran Associates Inc.，2022：23716-23736.

[17] SHOEYBI M，PATWARY M，PURI R，et al. Megatron-LM：training multi-billion parameter language models using GPU model parallelism [EB/OL]. (2020-03-13) [2024-12-26]. https://arxiv. org/abs/1909. 08053.

[18] OpenAI. GPT-4 technical report [EB/OL]. (2024-03-04)[2024-12-28]. https://arxiv. org/abs/2303. 08774.

[19] 巨涛，赵宇阳，刘帅，等. 面向图片识别的深度学习模型并行优化方法 [J]. 西安交通大学学报，2023，57(1)：141-151.
JU Tao，ZHAO Yuyang，LIU Shuai，et al. A parallel optimization method of deep learning model for image recognition [J]. Journal of Xi'an Jiaotong University，2023，57(1)：141-151.

[20] 巨涛，刘帅，火久元，等. 深度神经网络模型并行自适应计算任务调度方法 [J]. 吉林大学学报(工学版)，2024，54(12)：3601-3613.
JU Tao，LIU Shuai，HUO Jiuyuan，et al. Adaptive scheduling of computing tasks for deep neural network model parallelism [J]. Journal of Jilin University(Engineering and Technology Edition)，2024，54(12)：3601-3613.

[21] 巨涛，刘帅，王志强，等. 深度神经网络模型任务切分及并行优化方法 [J]. 北京航空航天大学学报，2024，50(9)：2739-2752.
JU Tao，LIU Shuai，WANG Zhiqiang，et al. Task segmentation and parallel optimization of DNN model [J]. Journal of Beijing University of Aeronautics and Astronautics，2024，50(9)：2739-2752.

[22] LIANG Feng，ZHANG Zhen，LU Haifeng，et al. Resource allocation and workload scheduling for large-scale distributed deep learning：a survey [EB/OL]. (2024-06-12) [2024-12-28]. https://arxiv. org/abs/2406. 08115.

[23] 杨紫超，吴恒，吴悦文，等. 基于性能建模的深度学习训练任务调度综述 [J]. 软件学报，2025，36(4)：1570-1589.
YANG Zichao，WU Heng，WU Yuewen，et al. Survey on task scheduling of deep learning training based on performance modeling [J]. Journal of Software，2025，36(4)：1570-1589.

[24] RADFORD A，KIM J W，HALLACY C，et al. Learning transferable visual models from natural language supervision [C]//Proceedings of the 38th International Conference on Machine Learning. Chia Laguna Resort，Sardinia，Italy：PMLR，2021：8748-8763.

[25] LI Junnan，LI Dongxu，XIONG Caiming，et al. BLIP：bootstrapping language-image pre-training for unified vision-language understanding and generation [C]// Proceedings of the 39th International Conference on Machine Learning. Chia Laguna Resort，Sardinia，Italy：PMLR，2022：12888-12900.

(编辑 刘杨 陶晴)