doi:10.11887/j.issn.1001-2486.25120014 http://journal.nudt.edu.cn

面向国产平台的大模型训练显存优化技术

李东升[1,2],唐 宇[1,2],乔林波[1,2*],吕倩茹[1,2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 国防科技大学 并行与分布计算全国重点实验室,湖南 长沙 410073)

摘 要:当前大模型训练中,模型参数量呈指数级增长与GPU显存容量缓慢增长的矛盾日益加剧。重计算和计算卸载两种显存优化技术,均是以时间换空间的思路来减小显存开销。本文分析了重计算技术和计算卸载技术的发展动态,针对国产人工智能计算平台的架构特点,剖析了国产平台上大模型训练显存优化面临的硬件带宽瓶颈、定制化指令集与软件生态适配等难题,分析了国产平台软硬件协同显存优化等技术发展路径,对MT-3000等国产计算平台上的大模型训练显存优化技术展开探讨,以期为国产平台上的大模型训练提供参考。

关键词:显存优化;重计算;计算卸载;国产平台;带宽受限
中图分类号:TP302.7 **文献标志码**:A **文章编号**:1001-2486(2026)02-284-12

Technologies for memory optimization for large model training on domestic platforms

LI Dongsheng[1,2], *TANG Yu*[1,2], *QIAO Linbo*[1,2*], *LYU Qianru*[1,2]
(1. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China;
2. National Key Laboratory of Parallel and Distributed Computing, National University of Defense Technology, Changsha 410073, China)

Abstract: In the current landscape of large-scale model training, the contradiction between the exponential growth of model parameters and the slow increase in GPU memory capacity has become increasingly prominent. Among memory optimization technologies, recomputation and computational offloading reduce GPU memory overhead by trading time for space. The development trends of recomputation and computational offloading were analyzed in this article. Then, the hardware bandwidth bottlenecks and software ecosystem adaptation challenges faced by memory optimization were analyzed, with a focus on the heterogeneous architecture characteristics of domestic artificial intelligence platforms. It also delved into the memory optimization technologies for large model training on domestic platforms such as MT-3000, with the aim of providing technical references for large model training on domestic platforms.

Keywords: memory optimization; recomputation; computational offloading; domestic platform; limited bandwidth

近年来,人工智能(artificial intelligence, AI)大模型的发展表明,将深度神经网络(deep neural network, DNN)模型参数量增加至千亿甚至万亿规模,会使大模型性能实现持续跃升。然而,大模型的快速迭代使得智能计算系统面临的“显存墙”问题更加凸显,如图1[1]所示,2016—2025年间,Transformer[2]类模型参数量从1亿级跃升至万亿级,增幅近10 000倍,而同期图形处理器(graphics processing unit, GPU)显存容量仅从P100对应的12 GB提升至A100/H100对应的80 GB,增幅不足7倍[1]。这种显著的“存储-算力”鸿沟带来了三方面的技术挑战:

1)显存容量瓶颈:单卡显存已无法承载万亿参数模型的激活值存储需求,模型训练所需显存和硬件提供的实际显存的差距持续扩大,硬件资源的供给能力难以匹配大模型训练需求,显著提

收稿日期:2025-12-06
基金项目:国家自然科学基金资助项目(62025208,62421002)
第一作者:李东升(1978—),男,安徽桐城人,研究员,博士,博士生导师,E-mail:dsli@nudt.edu.cn
*通信作者:乔林波(1988—),男,重庆人,副研究员,博士,硕士生导师,E-mail:qiao.linbo@ nudt.edu.cn
引用格式:李东升,唐宇,乔林波,等. 面向国产平台的大模型训练显存优化技术[J]. 国防科技大学学报, 2026, 48(2): 284-295.
Citation: LI D S, TANG Y, QIAO L B, et al. Technologies for memory optimization for large model training on domestic platforms[J]. Journal of National University of Defense Technology, 2026, 48(2): 284-295.

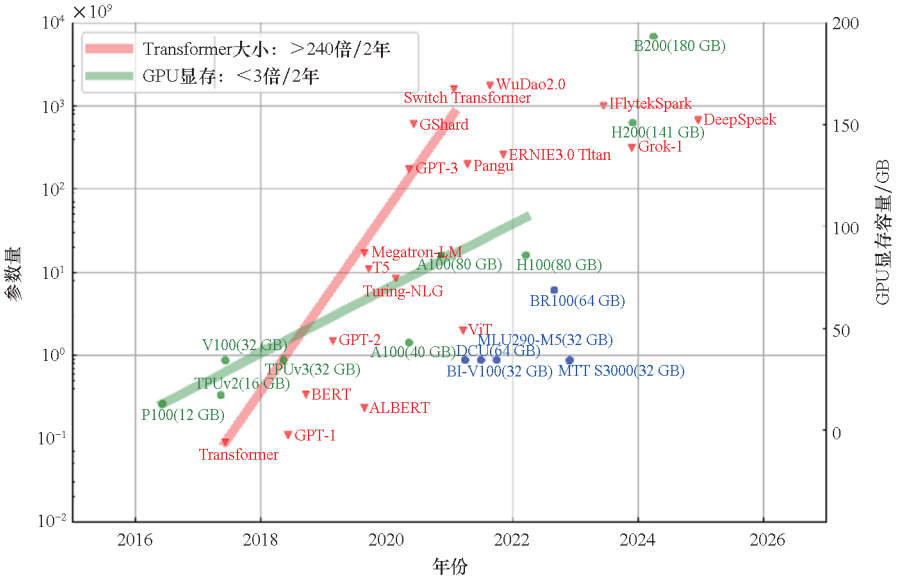

图 1　大模型参数量与 GPU 显存容量的发展趋势[1]

Fig. 1　Development trend of the model's parameter counts and the GPU's memory capacity[1]

升了大模型的研究探索成本。

2)访存效率困境:近年来人工智能大模型的计算需求每年增长约 10 倍,而动态随机存取存储器(dynamic random access memory,DRAM)访存带宽每 2 年仅增长约 1.6 倍,片间互连带宽(peripheral component interconnect express,PCIe)增速更只有每 2 年约 1.4 倍,访存带宽和互连带宽的增长速率不匹配,严重制约了大模型训练的效率并提高了训练成本。

3)效率-容量权衡难题:模型训练中数据并行批量增大虽可提升计算效率,但受限于随机梯度下降算法的稳定性,数据并行的批量大小被约束在一定区间,难以通过单纯增大批量缓解显存压力。

大模型训练的显存优化技术,是一类面向大规模深度学习模型训练场景,通过对 GPU 显存占用过程进行精准调控,缓解上述技术挑战的核心支撑技术。尽管中央处理器(central processing unit, CPU)内存优化和 GPU 显存优化在某些方面相似,但是 GPU 显存优化具有一定的独特性,已有的 CPU 内存优化技术无法直接迁移至 GPU 平台并发挥等效作用。目前大模型训练显存优化技术的研究已取得阶段性的进展[3],如模型并行、张量并行[4]、专家并行[5]、重计算[6]和计算卸载[7]等一系列方案。本文将主要聚焦于重计算(re-computation)和计算卸载(offloading)这两种显存优化技术展开深入探讨。其中,重计算作为深度学习模型训练常见的显存优化方式,它在前向计算过程中,通过将自动微分中的激活值提前释放,在反向计算过程再将这些激活值重算回 GPU 的方式,降低模型训练的显存占用。计算卸载类似 CPU 内存优化中的换入换出策略,将计算张量换出到 CPU 内存或者硬盘等更大的空间,待相关张量需要参与计算时再将其换入 GPU 显存,从而达到减少训练过程中 GPU 显存占用的目标。

本文将围绕重计算和计算卸载两种技术,介绍大模型显存优化技术的发展,根据解决两种技术难题的方法进行分类总结,并对国产平台上的大模型显存优化技术展开进一步的探讨与分析。

1　大模型训练中的典型显存优化技术

1.1　智能模型训练的显存需求

深度神经网络模型一次训练迭代中包含三个步骤。①前向计算:计算神经网络算子获取其对应的中间隐藏层的计算值。②反向计算:运行反向算子来计算参数的梯度。③更新参数:执行优化算法以更新参数值。而模型的显存占用分为四个部分,分别为模型参数(parameter)、梯度(gradient)、优化器状态(model states)、激活(activation)。在前向计算过程中,前向算子会输出大量的中间计算结果,这些中间计算结果称为激活,需要存储在计算设备中,占据一定量的存储空间。当模型层数加深时,激活数量增加,占据大量的内存。这些中间计算结果作为反向算子的输入,需要存储在显存中,直到相应的反向算子计算完毕。

若采用 FP32 的数据格式和 Adam[8] 优化器

训练大模型，模型参数、梯度和优化器状态的显存占比为 1∶1∶2，1 个 FP32 的模型参数(4 B)会带来 16 B 的显存占用。激活的显存占用大小通常与数据批量大小有关，见表 1。从表 1 可以看出，激活会随着数据批量大小增加而增加，当数据批量增长时，激活的显存占用将会超过模型参数、梯度和优化器状态所占的显存。例如，国产大模型 DeepSeek[9] 含有 6 710 亿参数，若采用 32 位浮点数的数据格式进行存储，其模型参数的显存开销为 2 499.67 GB，其激活显存多达 34.2 TB，该数值远超其模型参数的显存开销。

表 1 模型结构参数量和激活大小

Tab. 1 Model structure parameter quantity and activation size

模型结构	参数量大小	激活大小
多层感知机	$8h^2$	$19bsh$
注意力层	$4h^2$	$11bsh+5abs^2$
Llama2	70 B	$(2+9L)bsh+bsv$

注：b 表示模型训练批次大小，s 表示大模型输入序列长度，h 表示隐藏层大小，a 表示模型注意力头数，L 表示模型层数，v 表示训练词表大小。

综上，当前主流 GPU 设备的显存容量已难以满足大规模深度学习模型训练的实际显存需求。显存优化技术的核心目标是在尽可能不引入额外计算开销或通信开销的前提下，最大限度降低模型参数、梯度、优化器状态与激活值在单张 GPU 显存中的占用比例，重计算技术与计算卸载技术是该研究领域中两类典型且有效的优化方案。

1.2 重计算技术

重计算技术遵循“时间换空间”原则，前向计算时仅存储关键激活，其余中间结果随算随弃；在反向计算中，基于关键激活值重新计算缺失的激活值，从而保证反向梯度求解的正确性。如图 2 所示，在前向过程中，从 GPU 或其他计算设备中释放的算子，在反向过程中需要经过重计算过程重新返回 GPU 或其他计算设备中以确保计算正确性。正是由于额外的重计算流程，该技术相比常规的模型训练迭代，会引入额外的前向算子计算开销。

重计算技术的发展面临着多重挑战。首先，深度学习模型的迭代升级速度加快，而图结构获取与表示方法的演进未能与之同步，这导致精确获取计算图以及每个算子节点的计算量和显存占用情况变得相当困难，进而使得重计算建模受到影响。其次，随着模型规模不断增加，重计算位置的可选搜索空间呈现阶乘式爆炸增长，求解最优解的难度显著增加，最优重计算位置的求解难度显著提升。此外，大模型中控制流算子的引入进一步加剧了计算图获取的复杂性，使得相关问题的解决面临更多障碍。这些挑战相互交织，制约了重计算技术的实际应用和进一步发展，亟待从理论方法和技术实现等多个层面探索更有效的解决方案。

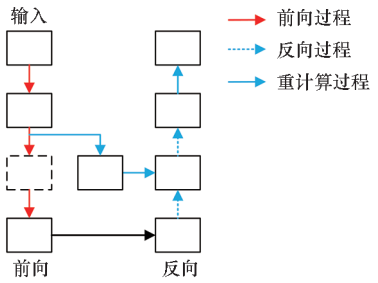

图 2 重计算示意图

Fig. 2 Illustration of recomputation

1.3 计算卸载技术

计算卸载技术是另一类主流的大模型显存优化方案，这种技术将暂时不参与计算的模型参数、梯度、优化器状态和激活从 GPU 显存卸载到其他存储设备上，待相关数据需要参与后续计算时，再重新加载回 GPU 显存，以降低显存的占用，如图 3 所示。计算卸载技术同样遵循“时间换空间”原则。

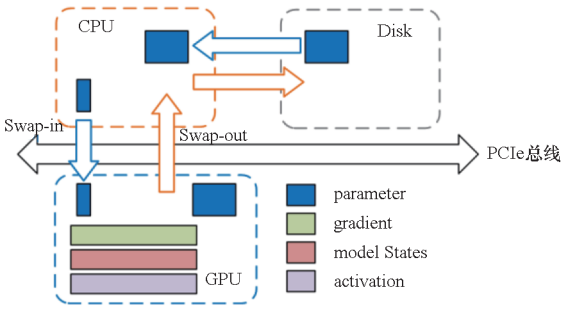

图 3 计算卸载示意图

Fig. 3 Illustration of computation offlading

在计算卸载技术的实际应用中，面临两类核心技术挑战。首先，PCIe 与 NVLink 在带宽速度层面存在显著差异，这一特性使得计算卸载过程中算子换入换出时机的选择变得相当复杂，需综合考虑数据传输效率与计算资源调度等因素，以实现传输与计算的高效协同。其次，计算卸载的异步执行依赖于数据流的有效配合。若缺乏合理的流管理机制，将难以实现计算任务与数据传输的高效重叠，进而影响整体训练性能。上述挑战既涉及硬件架构的特性适配问题，也关联软件调

度策略的优化设计,需从系统架构创新、编程模型改进及流调度机制优化等多个维度开展深入研究,以提升计算卸载技术在大模型训练场景下的实际应用效果。

2　重计算技术的发展

重计算技术的核心逻辑是在前向计算阶段选择性保留对反向梯度求解至关重要的激活值,其余非关键激活值则实时释放,这类被保留的关键激活值被定义为检查点。由此可见,重计算的本质是对检查点的选择进行规划。依据检查点规划问题的求解算法差异,重计算技术方案可以分为三类,分别是贪心与启发式算法方案、动态规划算法方案和数学规划方案。

2.1　贪心与启发式算法方案

贪心与启发式算法的核心优势在于能够快速生成检查点规划问题的可行解,但其固有缺陷是在复杂优化任务中极易陷入局部最优,难以获得全局最优解。revolve[10] 通过严谨的数学推导和证明,认为当满足以下约束时,可通过贪心算法获得最优的检查点规划方案:①每一个网络中的计算块具有相同的计算量和显存占用;②算子按序列顺序执行且之间没有依赖关系。然而,常见的神经网络架构通常难以满足上述约束条件,尤其是包含残差连接的网络模型,比如 ResNet[11],算子间的依赖关系显著,直接限制了该贪心算法的适用范围。Checkpoint 技术累计上一个检查点到目前计算算子的显存开销,超过阈值则设置检查点并重新积累显存的方案,快速规划检查点;动态张量重构[12](dynamic tensor rematerialization, DTR)通过设计启发式函数,为每一个张量计算优先分数,当显存超过阈值时,按照算子优先分数的次序释放算子。Mimose[13] 将神经网络模型中的每一个计算块进行两次前向计算,第一次前向计算过程中动态收集算子显存开销,第二次前向计算进行类似 Checkpoint 的检查点设置,遍历每一个计算块并完成对整个神经网络模型检查点的规划,解决了 DTR 动态检查点规划开销大的问题,实现了比 DTR 更高的吞吐量。MegTaiChi[14] 在 DTR 基础上结合了张量划分和自定义张量内存分配,一定程度上缓解了 DTR 内存碎片的问题。TWRemat[15] 使用树形分解计算图,将检查点设置问题转换成计算子图优化求解问题,能够在指定显存阈值下搜索出最优的检查点方案。

2.2　动态规划算法方案

动态规划算法将所有可能的检查点规划方案都进行遍历,以此达到全局最优解。checkpointVLAD[16] 使用分治求解的思想在无依赖的算子计算流中规划最优检查点设置。Rotor[17] 将计算图的操作视为一个序列的替换和组合,使用动态规划的方案找到最优的计算图操作序列。尽管动态规划算法提供了针对计算图的最优检查点规划方案,但对于大模型,动态规划算法求解耗时长,可长达数月,这一缺陷严重制约了其在大模型训练场景中的落地应用。

2.3　数学规划方案

数学规划方案通常使用整数线性规划(integer linear programming, ILP)或者混合整数线性优化(mixed integer linear programming, MILP)方法构建重计算中检查点规划的数学模型。在该模型中,每一个计算节点是否设置梯度检查点可视为一个 0/1 布尔变量,其优化目标是在显存约束下的模型经显存优化之后的最小计算时间。数学规划方案的显著优势是能够理论上获得全局最优解,但其不足之处在于算法复杂度高,求解耗时较长,当神经网络的规模达到千亿参数级别时,求解时间可长达数小时甚至数天。

Checkmate[18] 通过获取模型有向无环图,从而获取每一个节点的计算时间和显存消耗,为每一个节点设置一个是否检查点的布尔决策变量。使用数学规划工具如 Gurobi[19] 进行规划求解。Rockmate[20] 将 Rotor 方法和 Checkmate 的方法结合,将计算图划分为若干个计算块,在每一个计算块中使用整数线性规划算法进行最优检查点求解,有效提升了 Checkmate 的规划求解速度。Colossal-Auto[21] 将 Rotor 算法扩展,结合了计算卸载、重计算,同时考虑了通信时间,整体规划多级多卡时的检查点和计算卸载设置,为多级多卡环境下的重计算规划提供了参考。

2.4　重计算技术的突破节点与团队贡献

重计算技术的发展历程中涌现出多个具有里程碑意义的技术突破节点,不同研究团队的贡献推动着技术持续迭代升级。2016 年,Chen 等提出重计算技术首次将“时间换空间”思路应用于 DNN 训练[6],奠定重计算技术基础,解决了深层网络的激活值存储瓶颈;2020 年,华盛顿大学团队提出 DTR[12],创新引入启发式函数动态选择释放张量,突破了传统 Checkpoint 固定检查点的局限性,适配动态图场景;2020 年,加利福尼亚大学伯克利分校团队提出 Checkmate[18],首次将重计算建模为混合整数线性规划问题,实现全局最优

检查点规划,提升了重计算的理论精度;2022 年,北京航空航天大学团队提出 Mimose[13],通过“两次前向计算”动态收集算子开销,解决了 DTR 开销高的问题,提升了大模型场景的吞吐量。

3 计算卸载技术的发展

计算卸载技术的核心优化机制为,将暂时无须参与计算的数据从 GPU 显存换出至 CPU 内存、非易失性内存等辅助存储介质,待计算需求产生时再将其换入 GPU 显存,以此实现 GPU 显存占用的有效降低。在计算卸载技术中,数据换入与换出时间点的精准规划是决定技术优化效果的核心关键。与重计算技术的分类范式类似,按照求解数据换入与换出时间点规划问题的算法,计算卸载的技术方案可以分为两类:贪心与启发式算法和数学规划方案。

3.1 贪心与启发式算法方案

计算卸载的对象一般为模型参数,因此可以针对模型参数运算规则制定出相应的换入换出策略。例如,vDNN[22]面向卷积神经网络,针对模型参数设计贪心算法,在卷积操作后进行计算卸载,反向传播过程中进行重加载,结合统一计算设备架构(compute unified device architecture,CUDA)流实现 GPU 和 CPU 之间异步通信。SwapAdvisor[23]首先获取完整的有向无环图,之后针对有向无环图采用遗传算法[24]求解最优计算卸载的换入换出时间,不仅减少显存开销,而且吞吐率提升 53%。AutoSwap[25]定义启发式函数,对每一个张量打分,优先换出启发式函数分数高的张量,在必要时换入 GPU,从而实现自动计算卸载。TFLMS[26]依托 TensorFlow[27]框架,提出基于计算图模式匹配的卸载策略,通过对计算图中的典型算子组合模式进行识别,并在对应位置插入数据换入与换出节点,该技术突破了传统卸载方案的对象限制,可实现对任意类型张量的灵活卸载,有效降低了大规模模型训练的显存消耗。ZeRO-Offload[28]在 GPU 上面进行前向和后向计算,将梯度传给 CPU,并在 CPU 上进行参数更新,再将更新后的参数传给 GPU。为提高效率,ZeRO-Offload 将计算和通信并行起来,在反向传播阶段,将梯度值填满一个缓冲区后,计算新的梯度时将缓冲区内数据传输给 CPU,当反向传播结束,CPU 具有最新梯度值。同样地,CPU 在参数更新时也同步将已经计算好的参数传给 GPU。ZeRO-Infinity[29]在 ZeRO-Offload 的基础上,进一步将 CPU 内存中的参数卸载到非易失性内存(non-volatile memory express,NVMe)中,显著提高训练模型的参数量。PatrickStar[30]在 ZeRO-Offload 的基础上,考虑将张量形成的张量块进行卸载,并将张量块卸载到 CPU 内存和 NVMe 等异质存储介质中,增大了可训练的模型的参数量。

3.2 数学规划方案

数学规划方案针对计算卸载中数据换入换出时间点的规划问题构建数学模型,其核心决策变量为数据换入与换出的时间节点,优化目标设定为在满足 GPU 显存容量约束的前提下,实现模型整体训练时间的最小化。STRONGHOLD[31]结合 ZeRO 和混合整数线性规划,在卸载模型参数的同时卸载激活和梯度,进一步降低了 GPU 显存占用,为百亿甚至万亿参数规模的模型训练提供支持。FlexGen[32]使用混合整数线性规划计算卸载时间点,解决了在推理时的计算卸载规划。

3.3 计算卸载技术的突破节点与团队贡献

计算卸载领域的发展由多个关键的技术里程碑所组成。不同研究单位的持续探索为该领域注入了源源不断的动力,推动其在性能上不断突破上限。2016 年,麻省理工学院团队提出 vDNN[22],首次将张量卸载至 CPU 内存,开创计算卸载技术方向,解决了卷积神经网络的显存瓶颈;2021 年,微软团队提出 ZeRO-Offload[28],将梯度卸载至 CPU 并实现计算 - 通信并行,突破了千亿参数模型的训练限制;2021 年,微软团队升级提出 ZeRO-Infinity[29],引入 NVMe 存储扩展卸载空间,支持万亿参数模型训练;2023 年,斯坦福大学团队和加利福尼亚大学伯克利分校团队共同提出 FlexGen[32]卸载调度方案,有效提升了大模型的计算卸载方案规划效率。

4 重计算与计算卸载相结合的方法

重计算与计算卸载的融合技术,是当前大模型显存优化领域的重要研究方向,该类技术通过整合两种方案的核心优势,实现显存占用与训练效率的更优平衡。依据融合策略的求解算法差异,重计算与计算卸载相结合的技术方案可划分为启发式算法方案与数学规划方案两大类。

4.1 启发式算法方案

DELTA[33]采用和 DTR 类似的架构,在重计算基础上加入了计算卸载,该方案的核心突破在于打破了此前学术界普遍认为重计算与计算卸载结合时应优先以计算卸载为主的传统观点[34],为混合优化策略的设计提供了全新的技术思路。

MPress[35]则构建了更为全面的混合优化框架,将张量并行、流水并行、重计算与计算卸载四项关键技术进行深度融合,针对单机多卡的训练场景,充分发挥 NVLink 高速互连的技术优势,实现了计算与通信资源的高效调度,在同样的显存开销下该方案取得了优于零冗余优化方法[36](zero redundancy optimizer, ZeRO)的训练吞吐量。Moccasin[37]聚焦混合整数线性规划的求解效率优化,在传统混合整数线性规划模型的基础上,引入规划约束与拓扑约束两类关键约束条件,通过约束条件的限定大幅缩减可行解的搜索空间,最终实现求解速度的显著提升,在大规模计算图的优化场景中,其优化速度明显快于 Checkmate。

4.2　数学规划方案

重计算与计算卸载相结合的数学规划方案,其核心决策变量包含两类关键参数,分别为表征梯度检查点设置状态的布尔变量,以及定义数据换入换出时机的时间对变量,优化目标设定为在满足 GPU 显存容量约束的前提下,实现模型整体计算时间的最小化。XEngine[38]借鉴 Checkmate 的计算图建模思路,采用混合整数二次规划方法构建优化模型,实现重计算与计算卸载策略在异构计算与分布式计算设备上的协同调度,有效解决了异构与分布式场景下两种技术的联合设置难题。STR[39]在 Checkmate 的基础上,通过整合重计算与计算卸载的核心逻辑,引入混合整数线性规划、约束松弛条件与递归跟踪算法三大关键技术,构建了面向单机训练场景的混合方案,实现了显存占用与计算效率的精准平衡。OLLA[40]则提出了分阶段的优化策略,首先通过拓扑排序算法获取所有张量的完整生存周期信息,随后基于整数线性规划方法开展最小化张量驻留内存的方案规划,该方案在有效降低 GPU 显存占用的同时,还能够显著缓解内存碎片问题,进一步提升显存资源的利用率。

4.3　发展趋势

重计算与计算卸载融合技术的发展,离不开国内外研究团队的持续创新,其中多个关键技术突破推动了该领域的快速演进。例如,2023 年,中国科学技术大学团队提出 MPress[35],融合并行策略与混合方法,提升了单机多卡场景的吞吐量;2024 年,国防科技大学团队提出 DELTA[33]方案,针对动态控制流场景优化混合策略,突破了“以卸载为主”的传统认知;2022 年,德国人工智能研究中心提出 XEngine[38],针对软硬协同特性,优化混合整数线性规划求解速度。

整体来看,显存优化技术从“单一技术突破”(重计算或卸载的单一技术)向“多技术融合”(混合方法)演进,从“通用平台适配”向“专用平台定制”(国产平台针对性优化)深化,核心目标从“单纯节省显存”向“平衡显存、效率、硬件适配性”转变。国外团队聚焦通用平台的全局优化与规模化扩展;一些国内团队则聚焦国产平台的硬件异构特性(专用指令集、低 PCIe 带宽、异构存储),通过定制化算法与软硬协同,解决了国外技术在国产平台的适配难题,形成了差异化竞争优势。表 2从优化粒度、额外开销、国产平台适配性、适用模型规模及核心优劣势几个维度,系统对比了重计算、计算卸载及混合方法的技术特性与应用边界。各技术在实现机理与效能权衡上呈现显著的差异化特征。显存优化策略的选取需综合考量模型规模、硬件特性及工程落地成本,以实现最优的训练效能。

表 2　显存优化技术多维度对比

Tab. 2　Multi-dimensional comparison of memory optimization techniques

对比维度	重计算技术	计算卸载技术	混合方法
优化粒度	细粒度/粗粒度均可	粗粒度	细-粗粒度结合
额外开销	计算开销	通信/IO 开销	计算+通信开销
国产平台适配性	国产平台专用指令集增加算子适配难度	需适配国产框架设备管理接口	需定制化调度
适用模型规模	十亿级至千亿级	千亿级至万亿级	万亿级及以上
核心优势	部署成本低、不依赖总线带宽	显存节省显著、无计算误差	适配国产异构平台架构
核心劣势	动态控制流场景建模困难、国产平台指令集适配复杂	国产平台传输延迟高、异步调度复杂	开发门槛高、生态适配成本高

5　国产硬件平台

国产人工智能硬件平台是指由国内企业或科研机构自主研发,集成人工智能计算芯片、存储、

网络及配套软件的硬件系统,其核心构成有GPU、神经处理器(neural processing unit, NPU)、现场可编程门阵列(field programmable gate array, FPGA)等,旨在为AI训练与推理提供高效算力支持。

当前常见的国产人工智能计算平台有以下几种:

1)华为昇腾系列平台。华为昇腾系列平台基于华为自研的达·芬奇架构NPU构建,采用软硬协同优化设计。架构上采用"全栈自主"模式:硬件层搭载昇腾芯片,软件层与自研框架深度绑定,形成从云端训练到边缘推理的闭环生态。

2)寒武纪思元系列平台。寒武纪思元系列平台采用定制集成电路(application-specific integrated circuit, ASIC)架构的专用AI芯片路线,在专用NPU和多芯互联技术上形成核心优势,思元系列芯片支持芯片间200 GB/s双向带宽,可提升分布式训练效率。软件层面通过寒武纪NeuWare平台统一管理云端训练、边缘计算及终端设备,实现跨场景生态协同。

3)国防科技大学银河/天河系列超算和智算平台。国防科技大学银河/天河系列计算平台采用自主飞腾CPU、自主迈创加速器、自主互连网络与全栈软件生态的协同创新,服务于科学工程计算、人工智能与大数据处理等领域应用,其计算性能曾多次排名世界第一。

4)其他计算平台。如中国科学院曙光和海光系列平台(以海光x86架构通用处理器为基础算力载体,搭配曙光自研GPU加速卡形成"CPU + GPU"异构计算架构)、江南计算所神威系列系统(以自主申威众核处理器为核心,采用"片上多处理器 + 向量加速"的架构设计)、壁仞科技BR100[41]系列平台(采用"通用计算单元 + 专用AI加速模块"的异构设计,聚焦云端大模型训练与推理场景)等。

6 国产平台下大模型训练显存优化的关键问题及应对策略

重计算技术的核心要义在于检查点的智能规划,其高效实施依赖"计算图算子计算量均匀分布"与"存储层次访问延迟可预测"两大关键前提;计算卸载技术的核心目标在于高效数据迁移,其优化效果取决于"总线传输带宽"与"张量换入换出延迟"的精准可控;而重计算与计算卸载相结合的混合优化方法,则需实现时间开销与空间开销的动态平衡,其性能上限依赖"分层调度与软硬协同"的技术能力。随着国产人工智能平台的不断发展,其在架构设计、存储层次和软件生态等方面,与国外GPU平台存在显著差异[42],这些差异并非孤立存在,而是直接作用于前述优化技术的核心前提,既对现有技术方案的适配性形成针对性挑战,也为定制化优化策略的研发催生了独特机遇,进而对重计算与计算卸载技术在国产平台的落地应用提出全新要求。

6.1 现有研究的局限性

国产平台对显存优化策略的效能约束,本质源于存储层次、互联带宽、指令集等方面的硬件异构性与框架适配、编译优化等方面的软件生态成熟度两大核心变量。现有大模型显存优化研究虽已形成"算法 - 平台 - 应用"的初步框架,但在国产平台场景下,仍存在学术研究不充分、工程落地不彻底、生态协同不闭环三大核心局限性。

6.1.1 针对国产平台的学术研究不够充分

现有研究的理论短板体现在两个维度。一方面,现有国产硬件特性的抽象建模研究较少。当前主流研究多直接复用面向国外平台的计算图建模方法,例如基于CUDA架构的算子计算量与显存占用预测模型,针对国产平台的异构存储层次(如MT-3000的加速区HBM - 通用区DDR)、专用指令集(如鲲鹏SVE2向量扩展[43])构建专属的抽象模型的研究较少,导致显存优化规划方案易偏离国产平台下的最优解。另一方面,动态稀疏性模型与显存优化的融合仍处于空白阶段。混合专家模型(mixture of experts, MoE)等动态稀疏模型已成为大模型主流架构,但现有显存优化技术未针对"动态激活的专家参数"设计适配策略。国产平台的多级存储架构本可通过"激活专家参数驻留片上高带宽内存(high bandwidth memory, HBM)、未激活专家参数卸载至双倍数据率(double data rate, DDR)"提升效率,但现有研究多采用静态卸载/重算策略,导致动态场景下相关研究缺乏对应的理论支撑。

6.1.2 工程落地未能充分发挥国产平台硬件特性

现有优化方案多停留在应用层,例如基于PyTorch[44]、MindSpore[45]等深度学习框架的插件式开发,未深入编译器与硬件层进行定制化优化,导致硬件特性的效能潜力未能充分释放。具体而言,重计算的算子重算过程未利用国产芯片的专用计算单元,例如昇腾达·芬奇架构的矩阵计算单元[46],计算卸载的数据流调度未与编译器的算子融合、指令流水线调度机制协同设计,导致数据迁移过程与计算过程无法完全重叠,额外引入大

量的等待延迟。此外,国产平台对混合精度(FP16/FP8/INT8)的支持存在碎片化,如华为昇腾支持 FP16 但缺乏对 FP32 的高效支持[47],寒武纪思元系列平台则对 INT8 精度兼容性较好,同时支持 INT16 与 INT4 精度,对 FP16 精度的支持存在短板[48],而现有研究未针对这一精度特性优化显存策略,导致重计算在 FP16 精度下的梯度误差累积问题,影响模型训练的稳定性。

6.1.3　生态建设缺失适配标准与完善的工具链

国产人工智能计算平台的显存优化生态存在三大核心短板。第一,不同国产平台的显存优化接口碎片化,昇腾的 MindSpore 接口、寒武纪的 NeuWare 接口、壁仞的 BRAS 接口等互不兼容,导致同一显存优化方案需针对不同平台重复适配,大幅增加开发成本与维护难度。第二,缺乏统一的效能评估基准,当前尚未形成面向国产平台的标准化模型数据集与效能指标定义体系,不同优化方案的显存节省率、训练吞吐量等核心指标缺乏公平对比的依据。第三,工具链的监控与调试能力不足,国外有 DeepSpeed、Megatron-LM 等统一的显存优化框架,而国产平台的显存优化工具分散,如昇腾的 MindSpore 显存管理工具[49]和寒武纪的 Cambricon 显存管理工具[50]等均为平台专属工具,缺乏跨平台的统一框架,同时针对国产平台显存占用的精细化监控工具、重计算与计算卸载开销的量化分析工具存在明显缺失,导致优化策略的参数调优过度依赖经验,难以达到理论最优效能。

6.2　国产平台显存优化面临的挑战

6.2.1　硬件异构性导致优化策略适配难

国产加速器硬件层面普遍采用异构计算架构,以华为昇腾 910B、景嘉微 JM9 系列为代表的国产加速芯片采用异构计算架构,其显存控制器与 PCIe 接口为自主设计。在存储层次上,国产平台通常通过高带宽片上缓存和定制化内存子系统提升数据局部性。这种显存层次结构与国外产品存在差异,其 HBM 与板载双倍数据率同步动态随机存取存储器(double data rate synchronous dynamic random access memory, DDR SDRAM)的缓存层级设计不同,传统重计算中基于计算量均匀分割的检查点放置策略可能失效。此外,国产平台的 PCIe 接口实际传输效率较低,当计算卸载频繁交换大尺寸张量时,易导致计算卸载的“时间 - 空间”权衡失衡。

6.2.2　软件生态不完善制约自动化优化进程

软件层面,MindSpore、飞桨[51]等国产深度学习框架虽支持自动微分与计算图优化,但动态图模式下重计算的自动插桩能力较弱,计算卸载的设备管理接口与 CUDA 生态不兼容,需重新设计数据迁移策略。此外,国产平台在关键领域中的应用,常面临“算力受限”与“数据安全”的双重约束。如 MindSpore 静态图模式对动态控制流的捕获存在延迟,导致重计算的检查点规划无法准确建模算子计算量与显存占用情况。

6.2.3　专用指令集导致重计算算子建模困难

国产芯片厂商为提升计算效率,为龙芯、鲲鹏、飞腾等处理器引入了专属的定制化指令集,例如龙芯的 LoongISA[52]、鲲鹏的 SVE2 向量扩展、RISC-V 自定义指令等。这些专用指令集与硬件架构深度耦合,导致算子的计算过程高度依赖特定硬件的流水线调度与寄存器分配机制,显著增加了重计算算子的统一建模难度。在重计算过程中,激活值的重新计算需适配不同硬件的并行计算策略与资源竞争模型,无法通过统一的计算图抽象进行描述,导致基于通用计算图的重计算规划方案在国产平台上难以落地。

6.3　国产平台显存优化的定制化机遇

6.3.1　软硬件协同设计推动技术创新

针对国产固态硬盘(solid state drive, SSD)的低延迟特性,可设计专用计算卸载策略,如将重计算中高频访问的激活值缓存至 SSD 的单层单元(single level cell, SLC)工作模式区域,可大幅度降低从传统机械硬盘(hard disk drive, HDD)卸载数据或重加载的时间延迟,提升模型训练的吞吐率。此外,国产平台的 NPU 与 CPU 可形成异构集群,如昇腾 910B 的 CloudMatrix384 集群将集群中所有节点的 CPU DRAM 内存整合为共享的高性能内存池,这种内存池作为计算卸载的“中间层”,通过预取机制提前加载重计算所需激活值,减少 PCIe 往返次数[53]。

6.3.2　自主生态构建端到端优化体系

国产编译框架如华为 MindSpore 编译器可将重计算与计算卸载策略嵌入图优化阶段,在算子融合过程中自动识别可卸载张量,并生成基于国产 PCIe 接口的异步卸载指令。开源社区方面,OpenI 启智社区的“大模型显存优化”专项已积累大量适配国产平台的重计算脚本,其中针对昇腾平台的重计算方法在盘古大模型[54]训练中减少了峰值显存消耗。昇腾 MindSpore、海光 DCU + PyTorch 等开源软件生态支持底层接口的定制化开发,便于研发人员设计面向国产平台的重计算 - 卸载联合调度器,例如基于寒武纪 BANG 语

言开发的张量生存期分析工具,可精准评估张量的卸载优先级,提升计算卸载的优化效率。此外,国产平台对异构存储的良好支持,为 ZeRO-Infinity 等大规模显存优化方案提供了本土化落地路径,可支撑万亿参数规模大模型的高效训练。

6.3.3 构建多指令集统一抽象层及适配框架

针对不同国产芯片的专用指令集特性,例如向量长度、并行粒度、寄存器分配规则、运算单元适配范围等,提炼共性计算能力与差异化约束,形成统一的指令集能力描述框架。该模型不依赖具体硬件细节,而是将指令集的"计算效率、并行度上限、资源开销"等核心属性抽象为标准化参数,使重计算算子建模无须针对单一指令集重复设计,仅通过调用抽象模型即可适配多类国产芯片,降低建模复杂度。抽象层内置"指令集适配中间件",预先整合龙芯 LoongISA、鲲鹏 SVE2、寒武纪 BANG[55] 等专用指令集的张量运算特性,提供统一的算子计算量与显存占用预测接口。针对重计算核心算子(如激活值重算、梯度生成相关算子),根据目标芯片的指令集抽象参数,实现自动选择最优并行策略(如 SIMD 向量并行、线程级并行)、寄存器分配方案与指令组合方式,并生成硬件原生支持的算子代码。

6.4 未来发展方向

国产平台显存优化工作的未来发展趋势呈现多维度协同演进的态势,其核心在于通过软硬协同设计、异构计算架构创新及全栈生态建设,以应对大模型时代显存带宽、容量与效率的严峻挑战。具体而言,可归纳为以下几个方面:

1)国产智能计算平台上软硬协同优化的显存管理。国产芯片厂商正通过定制化指令集与硬件特性深度适配显存优化算法。例如华为昇腾 910C 通过达·芬奇架构的矩阵计算单元与动态编译技术解耦显存分配与模型编译流程,提升了长序列任务的显存访问效率。此类硬件原生支持能力结合编译器层的联合优化,可显著减少数据通信开销。

2)动态稀疏性与负载均衡技术的系统级创新将提升显存利用率。MoE 中专家动态激活引发的显存访问稀疏性,要求硬件调度与算法策略深度协同。此类技术尤其适配国产平台多级存储特性,可缓解因显存带宽不足导致的算力闲置问题。

3)开源生态共建与全栈工具链集成是国产平台破局的关键推力。当前国产 GPU 在生态兼容性上存在显著短板,但是 DeepSeek 等开源社区正提供替代路径。未来需构建覆盖开发、部署、监控的全栈工具链。此类工具链将降低开发者迁移成本,加速国产硬件与优化算法的落地应用。

总体来看,国产平台显存优化已从单一技术点突破转向系统级协同创新。通过芯片层指令定制、存储介质革新、动态调度算法、开源工具链赋能及场景化适配的深度融合,构建兼顾效率与成本的显存管理范式,将为国产 AI 算力自主化提供核心支撑。

6.5 技术初探:MT-3000 平台上的显存优化

MT-3000[56] 作为一款高性能的国产异构多区处理器,其独特的硬件架构为显存优化提供了创新基础,同时也面临异构资源协同、软件生态适配等挑战。MT-3000 采用通用区与加速区双分区设计,显存资源呈现三级分层结构:加速区集成 8 GB高带宽显存,通用区配备 64 GB 内存,并通过片上互联总线实现跨区数据交互。加速区计算任务需与 DDR 交换数据,但是加速区 HBM 与通用区 DDR 间的带宽不对称,导致计算卸载时数据迁移耗时占比较大,尤其在长序列 Transformer 训练中,激活值需在加速区计算后回传至通用区存储,单次迁移延迟较大。通用区 CPU 与加速区 NPU 的指令集差异,使重计算算子的硬件行为建模较为困难。此外,生态适配成本较高,PyTorch 等框架缺乏原生支持,需手动管理加速区显存分配,增加了开发复杂度。

基于以上问题,在 MT-3000 平台上的显存优化需采用分层优化方案。首先,采用显存分级管理策略,将高频访问的激活值保留于高带宽显存,通过最近最少使用(least recently used, LRU)等策略动态淘汰低频数据;实现检查点智能选址,基于计算图拓扑分析,将检查点设置在加速区内部存储边界,避免高计算复杂度算子回传通用区,降低迁移延迟。然后,实现软硬协同重计算优化,构建加速区指令 - 通用区指令的映射表,使重计算过程无须硬件感知,降低重计算过程对硬件的依赖,依据张量重算开销、显存占用及时效性三维指标,自动选择释放对象。最后,采用混合精度流水线设计,前向计算在加速区以 FP16 精度执行,梯度生成后降精度回传通用区存储,反向传播时再升精度至 FP16 重算。结合异步流水线调度,进一步降低显存占用。

DELTA[34] 方案将显存优化问题建模为背包问题,结合重计算技术和计算卸载技术,利用定制的启发式函数快速且精确生成算子显存优化策

略。DELTA 方案的核心目标是:在 MT-3000 多级存储架构下,动态平衡“重计算的时间开销”与“卸载的传输开销”,最大化显存节省率的同时控制总训练延迟。首先,DELTA 将显存优化决策建模为带约束的 0 – 1 背包问题,求解目标为:

$$\min_{L_r\subset T, L_s\subset T} \sum_{t\in L_r} c_r(t) + \sum_{t\in L_s} c_s(t)$$

$$\text{s.t.} \sum_{t\in T\setminus(L_r\cup L_s)} m(t) \leqslant B$$

$$L_r \cap L_s = \varnothing$$

其中,每个张量 $t\in T$,$c_r(t)$ 和 $c_s(t)$ 分别是张量 t 的重计算和卸载的开销,$m(t)$ 是张量 t 的显存开销,L_r 和 L_s 分别是重计算张量和卸载张量集合,B 是训练设备的显存上限。DELTA 利用 MT-3000 的多流调度能力,在反向计算重算低优先级张量时,同步加载卸载至 DDR 的高优先级张量,实现“重算 – 加载”并行,隐藏传输开销。DELTA 在 MT-3000 国产平台上通过框架适配以及通用区与加速区数据交换等技术,实现了在国产平台上的应用。

此外,TSO[57] 方案聚焦重计算技术,深入探究重计算技术在不同框架执行模式下的痛点,采用动态规划算法对模型算子序列进行执行模式规划,大幅度提升重计算技术训练速度。TSO 将模型算子序列的执行模式规划建模为分段动态规划问题,核心是划分算子序列为若干子段,每个子段选择最优执行模式。由于 TSO 方案聚焦于重计算技术,避免 MT-3000 等国产平台上计算核心与加速设备之间的数据交互,可以在国产平台上应用。TSO 规避跨区数据交互,所有重算操作均在 MT-3000 的加速区完成,无须与通用区交互。DELTA 方案通过重计算与卸载协同,更适合显存瓶颈突出的超大模型训练;TSO 方案则凭借低额外开销,更适合对训练速度敏感的大模型场景,两者均通过针对性的硬件适配,充分释放 MT-3000 的算力潜力。

重计算技术方案与计算卸载技术方案作为显存优化的核心手段,可与国产平台既有的并行计算、分布式计算技术深度融合。通过协同调度计算能力与存储资源,实现全栈式性能优化,显著提升智能模型在国产化平台上的训练推理效率。此类联合优化范式将进一步加速高算力需求场景的产业落地进程。

7　总结

本文围绕大模型训练中的“存储 – 算力”鸿沟,系统梳理了重计算与计算卸载两大核心显存优化技术的演进脉络,重点剖析了华为昇腾、寒武纪思元、MT-3000、壁仞 BR100 等典型国产计算平台的架构特性,阐明了硬件架构差异对显存优化技术适配性的深层影响,明确了“硬件异构导致策略适配难、软件生态不完善制约自动化、专用指令集增加建模复杂度”三大核心挑战,同时挖掘了“软硬协同设计、自主生态构建、多指令集统一抽象”三大定制化机遇,并通过 MT-3000 平台的分层优化方案与 DELTA、TSO 等技术的实践验证,形成了“技术特性 – 平台适配 – 效能优化”的论证链条。本文的研究价值体现在两个维度:一方面,从理论层面系统剖析了国产计算平台显存优化的独特挑战与适配规律,填补了现有研究多聚焦国外通用平台、对国产平台特性关注不足的空白,丰富了大模型显存优化技术的理论体系;另一方面,通过 MT-3000 平台的实践验证,提供了“数学建模 – 算法实现 – 硬件适配”的技术路径与量化数据,为国产芯片厂商、框架开发者及大模型训练从业者提供了直接参考。面向未来,大模型显存优化研究需以“破解国产平台核心痛点、释放自主算力潜力”为根本目标,从硬件定制化算法研发、全栈生态深度融合、动态稀疏场景适配、标准化评估体系构建等方向持续发力,突破大模型训练的“显存墙”约束,为万亿参数级大模型在国产算力平台的规模化训练提供核心技术支撑,最终助力我国人工智能产业实现自主可控、高质量发展。

参考文献(References)

[1] GHOLAMI A, YAO Z W, KIM S, et al. AI and memory wall[J]. IEEE Micro, 2024, 44(3): 33 – 39.

[2] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017: 6000 – 6010.

[3] TANG Y, QIAO L B, YIN L J, et al. Training large-scale language models with limited GPU memory: a survey[J]. Frontiers of Information Technology & Electronic Engineering, 2025, 26(3): 309 – 331.

[4] NARAYANAN D, SHOEYBI M, CASPER J, et al. Efficient large-scale language model training on GPU clusters using megatron-LM [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021: 58.

[5] FEDUS W, ZOPH B, SHAZEER N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity[J]. Journal of Machine Learning Research, 2022, 23(120): 1 – 39.

[6] CHEN T Q, XU B, ZHANG C Y, et al. Training deep nets with sublinear memory cost[EB/OL]. (2016 – 04 – 22)[2025 – 12 – 01]. https://arxiv.org/abs/1604.06174.

[7] 马玮良，彭轩，熊倩，等. 深度学习中的内存管理问题研究综述[J]. 大数据，2020，6(4)：56－68.
MA W L，PENG X，XIONG Q，et al. Memory management in deep learning：a survey[J]. Big Data Research，2020，6(4)：56－68.(in Chinese)

[8] KINGMA D P，BA J. Adam：a method for stochastic optimization[EB/OL]. (2017－01－30)[2025－12－01]. https://arxiv.org/abs/1412.6980.

[9] GUO D Y，YANG D J，ZHANG H W，et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning[J]. Nature，2025，645(8081)：633－638.

[10] GRIEWANK A，WALTHER A. Algorithm 799：revolve：an implementation of checkpointing for the reverse or adjoint mode of computational differentiation[J]. ACM Transactions on Mathematical Software (TOMS)，2000，26(1)：19－45.

[11] HE K M，ZHANG X Y，REN S Q，et al. Deep residual learning for image recognition[C]//Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)，2016：770－778.

[12] KIRISAME M，LYUBOMIRSKY S，HAAN A，et al. Dynamic tensor rematerialization[EB/OL]. (2020－03－18)[2025－12－01]. https://arxiv.org/abs/2006.09616.

[13] LIAO J J，LI M Z，SUN Q X，et al. Mimose：an input-aware checkpointing planner for efficient training on GPU[EB/OL]. (2022－09－06)[2025－12－01]. https://arxiv.org/abs/2209.02478.

[14] HU Z Z，XIAO J M，DENG Z Y，et al. MegTaiChi：dynamic tensor-based memory management optimization for DNN training[C]//Proceedings of the 36th ACM International Conference on Supercomputing，2022：1－13.

[15] KUMAR R，PUROHIT M，SVITKINA Z，et al. Efficient rematerialization for deep networks[C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems，2019：1359.

[16] SISKIND J M，PEARLMUTTER B A. Divide-and-conquer checkpointing for arbitrary programs with no user annotation[J]. Optimization Methods and Software，2018，33(4/5/6)：1288－1330.

[17] HERRMANN J，BEAUMONT O，EYRAUDDUBOIS L，et al. Optimal checkpointing for heterogeneous chains：how to train deep neural networks with limited memory[EB/OL]. (2019－11－27)[2025－12－01]. https://arxiv.org/abs/1911.13214.

[18] JAIN P，JAIN A，NRUSIMHA A，et al. Checkmate：breaking the memory wall with optimal tensor rematerialization[C]//Proceedings of Machine Learning and Systems，2020.

[19] Gurobi Optimization. Gurobi optimizer reference manual[EB/OL]. [2025－12－01]. https://docs.gurobi.com/projects/optimizer/en/current/index.html.

[20] ZHAO X Y，HELLARD T L，EYRAUD L，et al. Rockmate：an efficient，fast，automatic and generic tool for re-materialization in pytorch[EB/OL]. (2023－07－03)[2025－12－01]. https://arxiv.org/abs/2307.01236.

[21] LIU Y L，LI S G，FANG J R，et al. Colossal-Auto：unified automation of parallelization and activation checkpoint for large-scale models[EB/OL]. (2023－02－22)[2025－12－01]. https://arxiv.org/abs/2302.02599.

[22] RHU M，GIMELSHEIN N，CLEMONS J，et al. vDNN：virtualized deep neural networks for scalable，memory-efficient neural network design[C]//Proceedings of 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)，2016：18.

[23] HUANG C C，JIN G，LI J Y. SwapAdvisor：pushing deep learning beyond the GPU memory limit via smart swapping[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems，2020：1341－1355.

[24] ALHIJAWI B，AWAJAN A. Genetic algorithms：theory，genetic operators，solutions，and applications[J]. Evolutionary Intelligence，2024，17(3)：1245－1256.

[25] ZHANG J Z，YEUNG S H，SHU Y，et al. Efficient memory management for GPU-based deep learning systems[EB/OL]. (2019－02－19)[2025－12－01]. https://arxiv.org/abs/1903.06631.

[26] LE T D，IMAI H，NEGISHI Y，et al. TFLMS：large model support in TensorFlow by graph rewriting[EB/OL]. (2019－10－02)[2025－12－01]. https://arxiv.org/abs/1807.02037.

[27] ABADI M，BARHAM P，CHEN J M，et al. TensorFlow：a system for large-scale machine learning[C]//Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation，2016：265－283.

[28] REN J，RAJBHANDARI S，AMINABADI R Y，et al. ZeRO-Offload：democratizing billion-scale model training[C]//Proceedings of 2021 USENIX Annual Technical Conference，2021：551－564.

[29] RAJBHANDARI S，RUWASE O，RASLEY J，et al. ZeRO-Infinity：breaking the GPU memory wall for extreme scale deep learning[C]//Proceedings of SC21：International Conference for High Performance Computing，Networking，Storage and Analysis，2021：1－15.

[30] FANG J R，ZHU Z L，LI S G，et al. Parallel training of pre-trained models via chunk-based dynamic memory management[J]. IEEE Transactions on Parallel and Distributed Systems，2023，34(1)：304－315.

[31] SUN X Y，WANG W，QIU S H，et al. STRONGHOLD：fast and affordable billion-scale deep learning model training[C]//Proceedings of SC22：International Conference for High Performance Computing，Networking，Storage and Analysis，2022：1－17.

[32] SHENG Y，ZHENG L M，YUAN B H，et al. FlexGen：high-throughput generative inference of large language models with a single GPU[C]//Proceedings of the 40th International Conference on Machine Learning，2023：31094－31116.

[33] TANG Y，LI Q，YIN L J，et al. DELTA：memory-efficient training via dynamic fine-grained recomputation and swapping[J]. ACM Transactions on Architecture and Code Optimization，2024，21(4)：76.

[34] PENG X，SHI X H，DAI H L，et al. Capuchin：tensor-based GPU memory management for deep learning[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems，2020：891－905.

[35] ZHOU Q，WANG H Q，YU X Y，et al. MPress：democratizing billion-scale model training on multi-GPU servers via memory-saving inter-operator parallelism[C]//Proceedings of 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)，2023：556－569.

[36] RAJBHANDARI S，RASLEY J，RUWASE O，et al. ZeRO：

memory optimizations toward training trillion parameter models[C]//Proceedings of SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020: 1 - 16.

[37] BARTAN B, LI H M, TEAGUE H, et al. Moccasin: efficient tensor rematerialization for neural networks [EB/OL]. (2023 - 05 - 30) [2025 - 12 - 01]. https://arxiv.org/abs/2304.14463.

[38] SCHULER M, MEMBARTH R, SLUSALLEK P. XEngine: optimal tensor rematerialization for neural networks in heterogeneous environments [J]. ACM Transactions on Architecture and Code Optimization, 2022, 20(1): 1 - 25.

[39] WEN L J, ZONG Z, LIN L, et al. A swap dominated tensor re-generation strategy for training deep learning models[C]// Proceedings of 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022: 996 - 1006.

[40] STEINER B, ELHOUSHI M, KAHN J, et al. OLLA: optimizing the lifetime and location of arrays to reduce the memory usage of neural networks[EB/OL]. (2022 - 11 - 02) [2025 - 12 - 01]. https://arxiv.org/abs/2210.12924.

[41] HONG M K, XU L J. 壁仞™ BR100 GPGPU: accelerating datacenter scale AI computing [C]//Proceedings of 2022 IEEE Hot Chips 34 Symposium (HCS), 2022: 1 - 22.

[42] 许金伟, 王庆林, 李娅琳, 等. 多核数字信号处理卷积算法并行优化[J]. 国防科技大学学报, 2024, 46(1): 103 - 112.
XU J W, WANG Q L, LI Y L, et al. Parallel optimization of convolution algorithm on multi-core DSP [J]. Journal of National University of Defense Technology, 2024, 46(1): 103 - 112. (in Chinese)

[43] WEI H Y, LI W Q, SHEN S Y, et al. Vectorized SVE2 optimization of the post-quantum signature ML-DSA on ARMv9-A architecture [J]. Journalof Latex Class Files, 2021, 14(8): 1 - 11.

[44] PASZKE A, GROSS S, MASSA F, et al. PyTorch: an imperative style, high-performance deep learning library[C]// Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019: 721.

[45] CHEN L. Deep learning and practice with MindSpore[M]. Singapore: Springer, 2021.

[46] 李佳芯, 龚俊, 赵磊. 基于昇腾 AI 处理器的多路视频检测技术[J]. 兵器装备工程学报, 2025, 46(1): 258 - 266.
LI J X, GONG J, ZHAO L. Multi-channel video detection technology based on Ascend AI processor [J]. Journal of Ordnance Equipment Engineering, 2025, 46(1): 258 - 266. (in Chinese)

[47] LIAO H, TU J J, XIA J, et al. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing: industry track paper[C]//Proceedings of 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2021: 789 - 801.

[48] CHEN Z B, ZHENG F, YU Q, et al. Evaluating performance of AI operators using roofline model [J]. Applied Intelligence, 2022, 52(7): 7054 - 7069.

[49] WANG B C, YANG C Y, ZHU R, et al. Analysis of performance and optimization in MindSpore on Ascend NPUs[C]// Proceedings of 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), 2023: 1701 - 1708.

[50] XU X H, ZHU Y S. Optimizing logcumsumexp on cambricon MLU: architecture-aware scheduling and memory management[J]. International Journal of Applied Science, 2025, 8(3): 1 - 7.

[51] MA Y J, YU D H, WU T, et al. PaddlePaddle: an open-source deep learning platform from industrial practice[J]. Frontiers of Data and Domputing, 2019, 1(1): 105 - 115.

[52] HU W W, ZHANG Y F, FU J. An introduction to CPU and DSP design in China [J]. Science China Information Sciences, 2016, 59(1): 1 - 8.

[53] ZUO P F, LIN H M, DENG J B, et al. Serving large language models on Huawei CloudMatrix384 [EB/OL]. (2025 - 06 - 19) [2025 - 12 - 01]. https://arxiv.org/abs/2506.12708.

[54] ZENG W, REN X Z, SU T, et al. PanGu-α: large-scale autoregressive pretrained Chinese language models with auto-parallel computation[EB/OL]. (2021 - 04 - 26) [2025 - 12 - 01]. https://arxiv.org/abs/2104.12369.

[55] ZHOU R Y, LI Y J, ZHAO J C, et al. SYCL-MLU: unifying SIMT and SIMD in heterogeneous programming[J/OL]. CCF Transactions on High Performance Computing, 2025. [2025 - 12 - 01]. https://doi.org/10.1007/s42514 - 025 - 00252 - z.

[56] LU K, WANG Y H, GUO Y, et al. MT-3000: a heterogeneous multi-zone processor for HPC [J]. CCF Transactions on High Performance Computing, 2022, 4(2): 150 - 164.

[57] TANG Y, LI Q, ZHANG Y M. Boosting rematerialization training via execution mode splitting modeling on convex optimized dynamic programming[EB/OL]. [2025 - 12 - 01]. https://2025.eurosys.org/posters/final/eurosys25posters-final15.pdf.