面向边缘场景的高效大模型分层部署机制

An Efficient Hierarchical Deployment Mechanism for Large Models in Edge Scenarios

万卓然/Wan Zhuoran[1]，谢人超/Xie Renchao[1,2]，
王媛/Wang Yuan[3]，王芳/Wang Fang[3]，
唐琴琴/Tang Qinqin[1]，黄韬/Huang Tao[1,2]
（1. 北京邮电大学网络与交换技术全国重点实验室，中国 北京 100876；
2. 紫金山实验室，中国 南京 211111；
3. 中国电子科技集团公司第五十研究所，中国 上海 200063）
(1. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China;
2. Purple Mountain Laboratories, Nanjing 211111, China;
3. The 50th Research Institute of China Electronics Technology Group Corporation, Shanghai 200063, China)

DOI：10.12142/ZTETJ.202603006
网络出版地址：https://link.cnki.net/urlid/34.1228.TN.20260626.1024.008
网络出版日期：2026-06-26
收稿日期：2026-03-25

摘要：针对边缘设备资源受限导致大模型推理延迟高、吞吐量低的问题，提出了一种高效的分层部署优化机制，构建了“数据供给-空间保障-时间重叠”协同架构。具体而言，首先利用内存映射（mmap）实现权重按需加载以打破大模型加载的输入/输出瓶颈。其次通过静态内存预分配与指针偏移复用，彻底规避垃圾回收引发的系统抖动。最后提出基于权重矩阵粒度的流水线调度，拆解并行计算与加载任务，有效重叠显存访问延迟。实验表明，该机制在不增加显存占用的前提下显著提升了推理效率，推理吞吐量提升1倍以上，有效支撑了边缘侧大模型的实时推理需求。

关键词：边缘计算；大模型部署；推理优化；分层推理

Abstract: Addressing the issue of high inference latency and low throughput caused by resource constraints on edge devices, we propose an efficient hierarchical deployment optimization mechanism that establishes a collaborative architecture featuring “data supply-spatial assurance-time overlap”. Specifically, we first leverage memory mapping (mmap) to enable on-demand weight loading, thereby breaking through the input/output bottleneck associated with large model loading. Next, by statically pre-allocating memory and reusing pointer offsets, we completely eliminate system jitter caused by garbage collection. Finally, we introduce a pipeline scheduling approach based on the granularity of weight matrices, which decomposes parallel computation and loading tasks, effectively overlapping the latency of graphics processing unit (GPU) memory access. Experimental results demonstrate that this mechanism significantly improves inference efficiency without increasing GPU memory usage, more than doubling the inference throughput and effectively supporting real-time inference requirements for large models at the edge.

Keywords: edge computing; large model deployment; inference optimization; hierarchical inference

引用格式：万卓然, 谢人超, 王媛, 等. 面向边缘场景的高效大模型分层部署机制 [J]. 中兴通讯技术, 2026, 32(3): 30-36. DOI: 10.12142/ZTETJ.202603006

Citation：Wan Z R, Xie R C, Wang Y, et al. An efficient hierarchical deployment mechanism for large models in edge scenarios [J]. ZTE technology journal, 2026, 32(3): 30-36. DOI: 10.12142/ZTETJ.202603006

随着智能家居设备的逐步普及，家庭组网已演进为高度互联、数据密集的智能家居生态。在此背景下，大模型应用正加速从云端向边缘侧迁移[1-2]。然而，依赖云端应用程序编程接口（API）的集中式部署在工程落地中面临严峻挑战：在隐私层面，家庭敏感数据的云端传输极易引发泄露风险，且受制于严格的数据本地化监管刚性约束；在实时性层面，云端固有的网络延迟与拥塞难以满足跌倒告警、燃气处置等关键任务的快速响应需求，可能诱发安全隐患；在成本层面，高昂的云端算力租赁成本严重制约了该技术的规模化普及[3-4]。

为突破云端集中式处理在隐私安全、网络延迟及带宽成

基金项目：国家自然科学基金项目（92367104）

本上的固有局限，大模型部署的重心正在从云端向边缘侧迁移[5]。这一技术路径的转向，直接推动了边缘计算范式成为支撑家庭智能化转型的核心架构：边缘计算通过将计算能力下沉至距离数据源头最近的边缘设备，构建了一种具备数据本地化处理与快速响应能力的新型计算体系[6-7]。

然而，在面向大模型边缘部署时，现有主流框架往往难以兼顾显存效率与推理速度。Hugging Face Transformers[8]虽生态完善，但原生机制在处理长序列时显存碎片化严重。vLLM[9]虽通过 PagedAttention 实现了云端高吞吐推理，却因架构复杂难以适配边缘设备。llama.cpp[10]通过 INT4/INT8 量化实现大模型本地部署，但量化精度损失在复杂任务中较为明显。AirLLM[11]虽利用分层加载突破了显存瓶颈，但其串行处理机制导致推理延迟过高，仍无法满足边缘智能对实时性的严苛要求。

因此，针对上述显存效率与推理速度难以兼顾的挑战，本文提出了一套高效的边缘大模型分层部署机制：通过构建基于权重矩阵粒度的流水线调度模型，将加载与计算拆解为细粒度并行子任务，重叠显存访问延迟；设计了基于内存预分配的显存管理方案，利用静态预估与指针偏移复用，消除频繁垃圾回收（GC）开销；引入了内存映射（mmap）实现权重按需加载，解决了模型从外存到内存的输入/输出（I/O）瓶颈。所提方案在不增加显存占用的前提下大幅提升了推理吞吐量，有效解决了边缘大模型推理延迟过高导致的各类问题。

1 现有边缘大模型推理框架

面对边缘设备资源受限、算力与显存紧张的现实条件，大模型推理效率与部署可行性面临严峻考验。围绕这一问题，目前业界已提出多条典型技术路线与代表性推理框架，通过差异化方案攻克边缘部署痛点。

Hugging Face Transformers 是行业标准推理框架，其核心机制是依赖显存全量加载模型权重，采用静态显存管理策略，虽具备良好兼容性，但在边缘场景面临严峻挑战。比如，在长序列推理中，KV Cache 随上下文长度线性增长，极易触发内存溢出。显存容量成为部署超大模型的硬性瓶颈，限制了其在资源受限设备上的应用。vLLM 针对高并发场景引入 PagedAttention 机制，通过分页管理优化显存碎片化，显著提升吞吐量。然而，vLLM 高度依赖数据中心级算力与统一计算设备架构（CUDA）优化，复杂架构使得其在边缘侧难以高效运行，无法突破物理显存的绝对限制，部署灵活性差。

llama.cpp 另辟蹊径，采用生成式预训练变换器（GPT）生成统一格式（GGUF），将模型权重压缩至 4 bit 甚至更低精度，并结合 C/C++底层优化提升运算效率。这一策略大幅降低显存占用，使边缘设备得以承载更大模型。但量化带来的精度损失在逻辑推理、复杂语义理解等任务中尤为明显，模型输出质量与原生版本存在显著差异，因此需权衡资源与性能需求。AirLLM 则通过中央处理器–图形处理器（CPU–GPU）分层卸载技术实现“以时间换空间”，动态加载模型层以突破显存限制，使边缘设备可运行超大规模模型。然而，该策略的代价是极高的输入/输出（I/O）开销：频繁的主机–设备数据传输导致加载与显存清理耗时远超实际计算时间，推理延迟显著增加，实时性难以保障。

可以看出，现有主流方案各存在短板：静态加载框架（如 Transformers）受限于显存容量壁垒，难以部署大模型；量化技术（如 llama.cpp）以精度换资源，应用场景受限；动态卸载（如 AirLLM）虽突破显存限制，却因权重加载 I/O 阻塞导致延迟激增。究其根源，现有框架缺乏对推理全生命周期内存行为的动态优化，未能实现计算与内存调度的协同适配。边缘场景下的高效推理，亟需突破传统静态或粗粒度优化范式，构建能够实时感知资源状态、动态调度计算与内存访问的推理框架，在有限资源下实现性能与效率的平衡。这一方向成为当前边缘大模型部署的关键研究命题。

2 高效分层部署机制设计

针对现有分层卸载存在的显存加载延迟、显存清理开销、模型加载 I/O 瓶颈三大性能痛点，本文设计了一套包含映射加载、内存预分配与流水线调度的综合优化框架，旨在通过精细化的资源管理，实现推理过程的“零等待”与“低开销”。

2.1 整体架构设计

本文所述的 TriadFlow 架构由基于 mmap 的按需加载层、基于内存预分配的显存管理层以及基于权重矩阵粒度的流水线调度层构成，如图 1 所示。这 3 层在逻辑上呈现出“数据供给–空间保障–时间重叠”的递进与协同关系，共同构成了面向边缘场景的高效推理引擎。

在边缘设备有限的内存资源下，传统全量加载模式极易导致内存溢出。本文引入 mmap 技术，将庞大的模型权重文件直接映射至进程的虚拟地址空间。这一设计不仅消除了传统 I/O 读写过程中的数据拷贝开销，更实现了权重数据的“按需加载”。它为上层机制提供了逻辑上的“全量数据视图”，解决了超大模型在边缘侧“存不下”的根本矛盾，为后续调度提供了数据源头的供给保障。

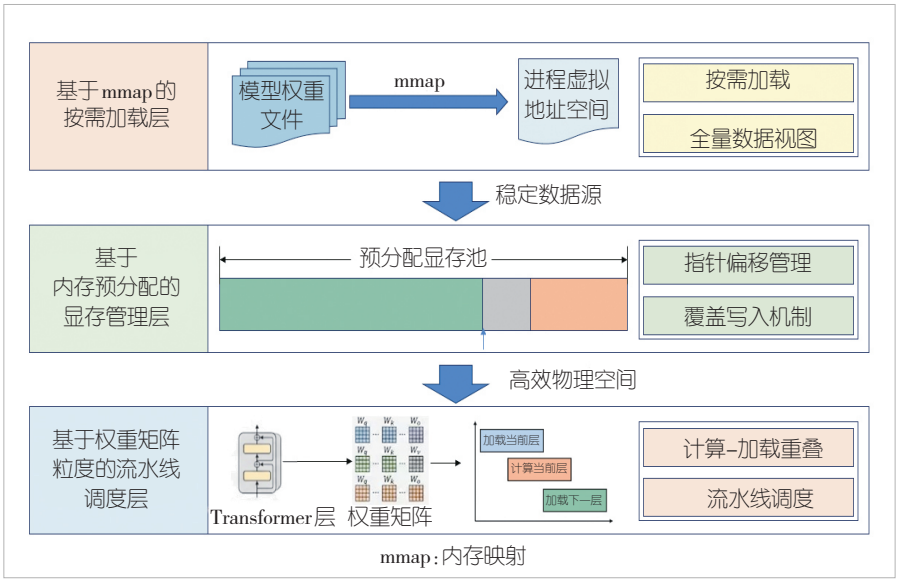


图1 TriadFlow整体架构设计

mmap虽然解决了数据的逻辑存在，但频繁的分层卸载仍需在物理显存中进行高频的数据交换。若采用传统的动态内存分配，频繁的申请、释放与GC将引入巨大的系统抖动。为此，本文设计了静态内存预分配方案。推理引擎在启动阶段即可根据计算图静态预估峰值显存需求，并预先向系统申请一块连续的内存构建“显存池”。通过指针偏移与覆盖写入机制，该方案彻底消除了运行时的动态内存管理开销，为流水线调度提供了稳定、无碎片的物理运行环境。

在具备了稳定的数据源（mmap）和高效的物理空间（预分配显存池）后，调度层致力于解决“计算-访存”失配问题。传统分层卸载以“层”为单位进行串行处理，导致计算单元在等待权重加载时处于空闲状态。本文将调度粒度从“层”细化至“权重矩阵”，将Transformer层拆解为独立的线性运算单元。调度器利用预分配显存池中的指针管理，精确控制数据搬运与计算任务的时序，使得当前层的计算与下一层的权重加载在时间上完全重叠。

综上所述，这3层架构紧密耦合：mmap加载机制为显存管理层提供了按需供给的数据流；显存管理层为流水线调度提供了无锁、低延迟的空间复用环境；而流水线调度层则充分利用前两者的优势，将显存访问延迟完全重叠在计算过程之中。三者协同，最终可在不增加显存占用的前提下，大幅提升边缘大模型的推理吞吐量。

2.2 基于mmap的模型加载加速

在边缘设备有限的内存资源约束下，传统全量加载模式不仅面临“存不下”的物理限制，更因冗余的数据搬运导致严重的I/O阻塞[12]。因此，本研究引入mmap机制，旨在通过底层系统调用的优化，构建一个逻辑上全量、物理上按需的权重加载层，为后续推理过程提供高效、稳定的数据源供给。

大模型权重文件通常高达数GB甚至数十GB，且多以二进制格式存储。传统的I/O读取方式采用“磁盘-内存”完整拷贝模式，需将完整的权重数据从磁盘分区读取至内存缓冲区，再搬运至物理内存的指定地址。这一过程不仅受限于磁盘I/O带宽，还极易因大量数据的集中拷贝导致内存碎片化。实验数据表明，7B模型的内存加载耗时为1.32 s，而72B模型则激增至112.28 s，直观反映出I/O带宽与模型参数量级之间的尖锐矛盾。此外，碎片化的内存空间无法被有效利用，易引发系统颠簸，导致加载过程频繁触发页交换，造成CPU与内存资源的无效消耗，严重影响边缘设备长时间运行的稳定性。

针对上述加载缓慢与内存碎片化问题，本文引入mmap机制，结合边缘设备的存储与内存特性对模型加载流程进行重构，并充分利用该机制的“文件-虚拟内存映射”特性，将大模型权重文件直接映射到推理进程的虚拟地址空间，具体实现过程如图2所示。首先，通过系统调用mmap()函数，将权重文件的磁盘地址与进程虚拟地址空间中的一段连续地址区间建立映射关系，该映射关系被记录在进程的页表中。其次，设置映射权限为“只读”，确保权重数据不被意外修改，同时避免写入操作带来的额外开销。最后，将映射后的虚拟地址区间与推理引擎的权重访问接口关联，使引擎可直接通过虚拟地址访问权重数据，无需经过传统I/O的数据拷

图2 mmap加载机制架构

贝流程。

这一机制使模型推理引擎能够像访问内存变量一样直接读取磁盘上的权重数据，真正实现权重数据的“按需加载”，其加载逻辑与推理流程深度绑定：当推理引擎执行到某一层计算且需要访问该层对应的权重数据时，系统会通过虚拟地址发起访问请求。此时，操作系统会检查该虚拟地址对应的物理内存页面是否已加载。若未加载，则触发页面置换机制，将磁盘上对应的权重数据块（通常为4 kB或8 kB，与系统页大小一致）换入物理内存，同时更新页表。若已加载，则直接从物理内存中读取数据，无须访问磁盘。未被访问的权重数据仍保留在磁盘中，无须占用物理内存资源，有效降低了边缘设备的内存压力。这种加载方式不仅能够大幅提升超大权重文件的加载效率，还可通过“按需加载”减少内存数据的冗余存储，有效规避内存碎片化对推理性能的影响。

基于mmap的按需加载方案不仅解决了模型权重在边缘侧的“存与取”问题，更为上层机制提供了逻辑上的全量数据视图。然而，仅有高效的数据供给尚不足以支撑高性能推理。若缺乏物理显存层面的精细化管理，频繁的内存分配与回收仍将引发系统级抖动。因此，下文将基于这种数据供给基础，进一步探讨如何通过静态内存预分配策略构建“无锁、低延迟”的物理运行环境。

2.3 基于内存预分配的显存管理优化

基于上述关于数据供给机制的论述，本节聚焦“空间保障”层面，核心目标是消除运行时显存管理的系统开销。尽管mmap解决了权重数据的逻辑存在，但在推理过程中，激活值与中间结果的高频动态变化，往往迫使系统进行频繁的显存申请与释放。针对这一问题，本文设计了轻量化的显存池管理器，摒弃传统的动态分配策略，转而采用静态预分配与指针偏移复用的机制，旨在从根源上切断由GC引发的性能抖动，为流水线调度提供稳固的物理空间支撑。

为避免显存溢出，现有分层加载框架需频繁执行显存释放与重新分配操作。这一过程涉及操作系统层面的锁竞争与碎片整理，引入了巨大的时间成本。例如，在14B模型推理中，显存清理耗时（12.84 s）已显著高于计算耗时（0.67 s），成为首要瓶颈。而72B模型的清理时间更是达到了60.48 s，约占用总耗时（302.44 s）的20%。频繁的显存管理不仅消耗时间，还加剧了系统资源竞争，降低了推理稳定性。

针对显存清理环节的频繁开销问题，本研究提出一种内存预分配优化方案，通过静态管理模式替代传统动态分配策略，从根源上减少显存回收与重新分配带来的系统损耗，如图3所示。

现有推理框架因无法提前预判推理过程中的内存需求波动，普遍采用保守的动态分配策略，即根据实时计算需求动态申请和释放显存空间，这一模式导致推理过程中充斥着大量显存回收、重新分配的系统调用，不仅增加推理延迟，还易引发显存碎片，尤其在边缘设备有限的显存资源下，碎片积累会进一步降低显存利用率。该优化方案的核心思路围绕“空间换时间”与“静态管理”展开。本文设计了一款轻量化显存池管理器，该管理器包含内存需求预估、显存区域划分、动态指针调度3个核心模块。在推理任务启动前，管理器先通过遍历模型计算图，统计各层权重、激活值、中间特征向量的最大占用空间，结合推理批次大小及边缘设备显存资源上限，精准计算出模型推理过程中的峰值内存消耗。系统据此提前预分配一块连续的、固定大小的显存区域，其容量预留5% ~ 10%的冗余空间，确保无需额外动态扩容，同时避免显存资源浪费。预分配的显存区域按功能类型可以划分为权重存储区、激活值存储区、中间结果存储区。各区域边界固定，通过指针标记实现独立管理，以避免不同类型数据的相互干扰。

在推理过程中，本方案采用指针动态管理机制替代传统内存回收机制，具体如图4所示。系统为预分配的各显存区域分别设置读写指针与空闲指针，通过移动指针标记显存空间的占用与释放状态，无须对显存块进行物理上的回收与重新分配。权重存储区采用“一次性加载、全程复用”的模式。权重数据加载后固定存储在对应区域，直至推理任务结束。激活值与中间结果存储区采用“覆盖写入”模式。当某一层计算完成后，后续层的激活值的计算结果直接覆盖前一层的闲置空间，通过指针偏移实现空间复用，彻底避免了频繁触发操作系统级的GC操作，从根本上规避频繁GC带来的性能损耗。

图3 预分配显存机制

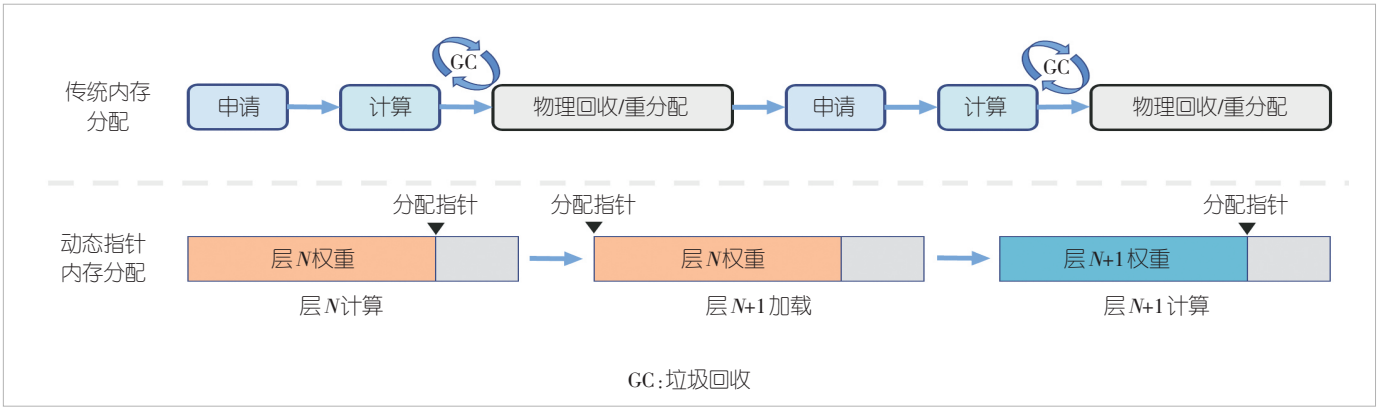


图4 动态指针内存分配机制

然而，仅有静态的空间保障仍无法解决“计算–访存”失配导致的时间效率问题。当显存访问延迟无法被重叠时，计算单元的空闲将直接导致吞吐量下降。因此，下文将围绕“时间重叠”，深入探讨如何通过细粒度的流水线调度，将显存访问延迟完全掩藏于计算过程之中。

2.4 基于权重粒度的流水线调度机制

综合前文所述的数据供给与空间保障机制，本研究构建了边缘推理引擎的基础架构。但要实现极致的推理效率，关键在于“时间重叠”维度的调度优化。在边缘设备有限的算力条件下，传统的粗粒度（层级别）调度往往导致计算单元在等待权重加载时处于空闲状态，形成显著的“气泡”延迟。针对该问题，本文提出了一种基于权重矩阵粒度的流水线调度机制。该机制深度结合Transformer模型，将计算与加载任务拆解为细粒度的并行子任务，旨在通过精确的时序控制，实现加载与计算的完全重叠，从而在不增加显存占用的前提下使硬件利用率达到最大。

现有分层加载框架以Transformer层为单位进行显存交换，导致计算单元在等待下一层权重加载时处于空闲状态，形成了显著的“计算–等待”间隙。虽然AirLLM采用了预取策略进行重叠计算和加载操作，但实验数据显示，加载操作耗时远长于计算操作，简单的层级预取没有办法实现时间覆盖，“计算–等待”间隙需要被优化。随着模型参数量级的增加，显存加载延迟呈非线性增长：7B模型的显存加载耗时为6.83 s，而72B模型则高达123.59 s——时长增加了近20倍，严重制约大模型边缘实时推理能力。

为解决显存加载引发的计算单元空闲问题，本工作摒弃传统“层”级加载粒度，提出基于“权重矩阵”粒度的流水线调度机制。该机制充分适配Transformer模型的内部计算特性，实现加载与计算任务的并行协同。

本工作深入剖析Transformer层[13]的内部计算结构，结合其“预归一化”或“后归一化”的典型架构，将单一层级完整拆分为多个独立且可并行调度的矩阵运算单元，具体拆分逻辑贴合Transformer的正向计算流程：首先，将自注意力机制模块拆分为查询（Q）、键（K）、值（V）3个独立的权重矩阵运算单元，三者均基于输入嵌入向量与对应权重矩阵完成线性变换，且运算过程相互独立，无依赖关系；其次，将自注意力输出后的线性投影层拆分为独立的权重矩阵运算单元，负责对注意力加权后的特征向量进行维度映射与整合；最后，将前馈网络（FeedForward）层拆分为两个串联的线性变换权重矩阵运算单元，分别对应“升维–降维”的计算过程，同时将层归一化模块的参数提取为独立的参数单元，确保其与矩阵运算单元的调度协同。在此基础上，本文明确各拆分单元的计算时序与显存需求，例如Q、K、V权重矩阵的运算可同步启动，其显存加载需求均为固定维度的浮点数矩阵。而FeedForward层的两个线性变换单元需按“先升维后降维”的顺序依次运算，显存加载需遵循对应时序。优化后的调度器打破“整层加载完成后再启动计算”的传统逻辑，将显存加载任务拆解为与计算任务同步推进的子任务流，形成“计算–加载”并行的流水线模式。在计算单元处理当前批次权重矩阵的运算任务时，数据搬运单元已根据预设时序，提前启动下一阶段所需权重矩阵的显存加载操作，确保计算单元完成当前任务后可立即调用新的权重数据，无需进行等待。

基于这种细粒度流水线设计，本文成功实现了显存加载时间与计算时间的最大程度重叠，有效重叠显存访问延迟，基本消除粗粒度加载模式下普遍存在的计算“气泡”等待时间。具体而言，Q、K、V权重矩阵的加载与运算可实现完全并行，如图5所示。当计算单元处理Q矩阵运算时，数据搬运单元同步加载K、V矩阵；待Q矩阵运算完成后，K、V

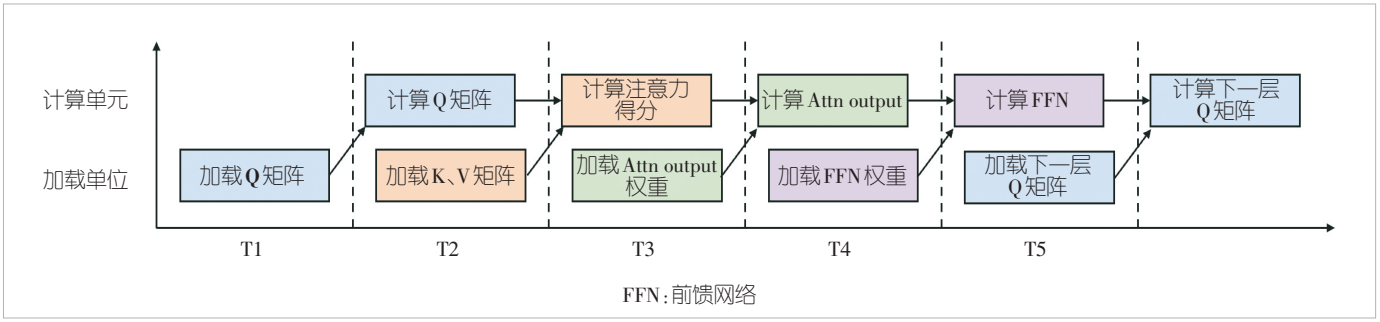


图5 流水线加载重叠显存加载时间与计算时间

矩阵加载完毕，可立即启动注意力得分计算。自注意力输出的线性投影层权重加载，可与注意力得分计算过程并行，进一步压缩无效等待时间。

该机制通过将调度粒度从“层”细化至“权重矩阵”，成功实现了显存加载时间与计算时间的最大重叠，大大减小了计算空泡。综上所述，本章提出的“数据供给—空间保障—时间重叠”三位一体优化框架：依托mmap破除I/O瓶颈，借助预分配消除了GC抖动，采用细粒度流水线重叠访存延迟。这3层机制紧密耦合、互为支撑，共同构成了面向边缘场景的高效大模型分层部署框架。

3 实验验证

3.1 实验设置

为验证本文提出的高效分层部署机制在边缘多智能体场景下的有效性，实验选取了典型的边缘计算设备作为测试平台，并对比了优化前后的推理性能，具体如表1所示。

3.2 性能对比分析

如图6所示，通过引入基于权重粒度的流水线调度与内存预分配优化，本工作提出的TriadFlow成功在单Token推理耗时方面较基准组原版AirLLM的平均耗时在不同尺寸的模型上都有巨大降低，平均降低达到53%。在较小模型上，降低效果更佳，7B模型降低达64%。通过消除显存加载等待与内存清理开销，推理吞吐量提升了1倍以上。

此外，在显存占用方面，如图7所示，本工作提出的TriadFlow保持了与基准组AirLLM相当的低显存占用水平，均维持在4 GB以下。这说明本文提出的优化策略并未引入额外的显存负担，依然适配边缘设备的资源限制。

表1 实验条件设置

硬件环境	模型配置	对比对象
采用NVIDIA L40（48 GB RAM），使用torch.cuda.set_per_process_memory_fraction限制显存6 GB，模拟家庭边缘算力环境	选用Qwen2.5-7B、Qwen2.5-14B、Qwen2.5-32B和Qwen2.5-72B[14-15]模型作为基准模型	基准组为原始AirLLM框架，实验组为本文提出的TriadFlow

RAM：随机存取存储器

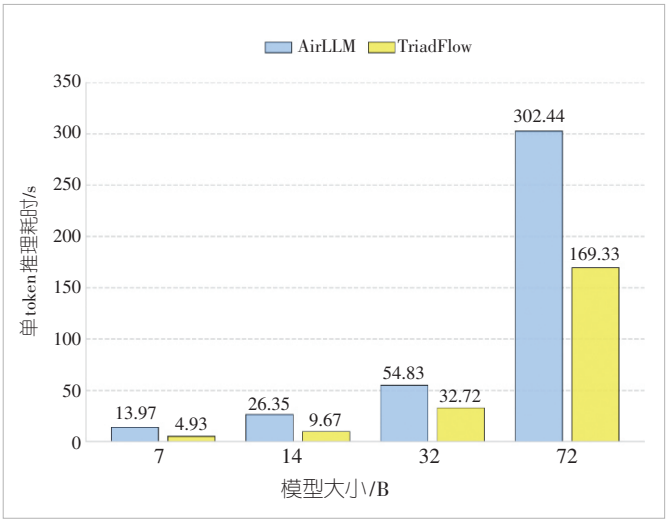


图6 单Token推理平均耗时对比

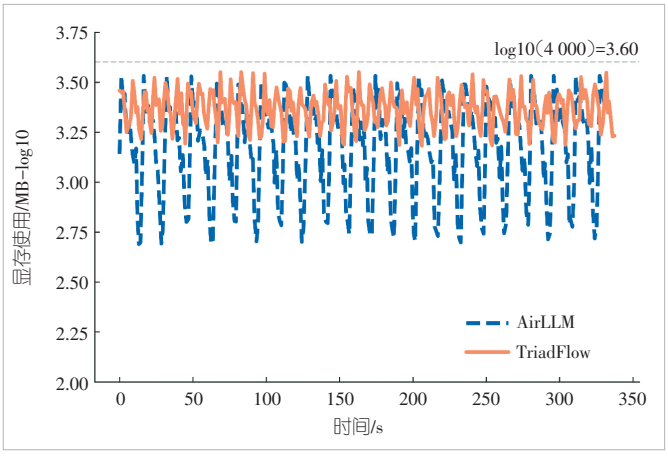


图7 显存占用对比

4 结束语

针对边缘大模型推理延迟过高的问题，本文提出了一种高效分层部署优化机制。通过剖析现有框架的耗时构成，明确显存加载、内存管理与模型I/O是导致延迟的主要原因，并设计了3项核心优化策略：基于权重粒度的流水线调度机制有效重叠了显存访问延迟，内存预分配策略消除了频繁的显存回收开销，mmap内存映射机制则解决了大模型加载的I/O瓶颈问题。实验结果表明，所提机制在不增加显存占用的前提下，将推理延迟降低50%以上。未来的工作将探索该机制在异构边缘设备集群中的分布式扩展，并研究如何结合量化技术进一步提升推理效率。

参考文献

[1] Li Z H, Li T, Feng W J, et al. PRIMA. cpp: fast 30-70B LLM inference on heterogeneous and low-resource home clusters [PP/OL]. arXiv (2025-04-07) [2026-04-16]. https://arxiv. org/abs/2504.08791

[2] Sarhaddi F, Nguyen N T, Zuniga A, et al. LLMs and IoT: a comprehensive survey on large language models and the Internet of Things [J]. IEEE open journal of the communications society, 2026, 7: 3585-3616. DOI: 10.1109/ojcoms.2026.3680064

[3] Lin Z, Qu G Q, Chen Q Y, et al. Pushing large language models to the 6G edge: vision, challenges, and opportunities [J]. IEEE communications magazine, 2025, 63(9): 52-59. DOI: 10.1109/MCOM.001.2400764

[4] Huang X Y, Shen L M, Ma Z J, et al. Toward privacy-preserving and personalized smart homes via tailored small language models [J]. IEEE transactions on mobile computing, 2026, 25(7): 9505-9520. DOI: 10.1109/tmc.2026.3654976

[5] Shi W S, Cao J, Zhang Q, et al. Edge computing: vision and challenges [J]. IEEE Internet of Things journal, 2016, 3(5): 637-646. DOI: 10.1109/JIOT.2016.2579198

[6] 王睿, 齐建鹏, 陈亮, 等. 面向边缘智能的协同推理综述 [J]. 计算机研究与发展, 2023, 60(2): 398-414

[7] 张晓东, 张朝昆, 赵继军. 边缘智能研究进展 [J]. 计算机研究与发展, 2023, 60(12): 2749-2769. DOI: 10.7544/issn1000-1239.202220192

[8] Wolf T, Debut L, Sanh V, et al. Transformers: state-of-the-art natural language processing [EB/OL]. [2026-04-14]. https://github.com/huggingface/transformers

[9] Zhu W, Li S, Zheng S, et al. vLLM: high-throughput LLM serving with PagedAttention [C]//Proceedings of the 29th Symposium on Operating Systems Principles. ACM, 2023: 1-16

[10] Gerganov G. LLaMA model inference in pure C/C++ [CP/OL]. [2026-04-15]. https://github.com/ggerganov/llama.cpp

[11] Lyogvin. AirLLM: efficient inference framework for large language models on low-resource GPUs: Version 2.11.0 [CP/OL]. [2026-04-15]. https://gitcode. com/GitHub_Trending/ai/airllm

[12] 高赫然, 吴恒, 许源佳, 等. 面向深度学习训练的内存交换机制综述 [J]. 软件学报, 2023, 34(12): 5862-5886

[13] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need [C]//Advances in Neural Information Processing Systems. ACM, 2017: 5998-6008. DOI: 10.48550/arXiv.1706.03762

[14] Qwen. Qwen2.5: a party of foundation models [EB/OL]. (2024-09-19)[2026-04-15]. https://qwenlm.github.io/blog/qwen2.5/

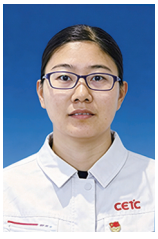
[15] Yang A, Yang B S, Hui B Y, et al. Qwen2 technical report [PP/OL]. arXiv(2024-07-15) [2026-04-15]. https://arxiv. org/abs/2407.10671

作 者 简 介

万卓然，北京邮电大学在读硕士研究生；主要研究方向为边缘大模型部署与推理任务调度。

谢人超，北京邮电大学教授、博士生导师；主要研究方向为算力网络、网络人工智能等。

王媛，中国电子科技集团公司第五十研究所工程师；主要研究方向为算力网络、软件定义网络等。



王芳，中国电子科技集团公司第五十研究所高级工程师；主要研究方向为软件定义网络、网络人工智能等。

唐琴琴，北京邮电大学副研究员、硕士生导师；主要研究方向为算力网络、网络人工智能、网络孪生等。

黄韬，北京邮电大学教授、博士生导师；主要研究方向为未来网络、确定性网络等；先后主持科技部、工信部等重大项目10余项；获中国通信学会科技奖、中国电子学会科技奖等20余次；发表论文120余篇，获授权专利100余项。