ZipServ：采用硬件感知无损压缩的快速、内存高效大语言模型推理

Ruibo Fan
香港科技大学（广州）中国广州
ruibo.fan@connect.hkust-gz.edu.cn

Xiangrui Yu
香港科技大学（广州）中国广州
xyu868@connect.hkust-gz.edu.cn

Xinglin Pan
香港科技大学（广州） 广州，中国
x
pan413@connect.hkust-gz.edu.cn

Zeyu Li
香港科技大学（广州） 广州，中国
zli755
@connect.hkust-gz.edu.cn

Weile Luo
香港科技大学（广州） 广州，中国
wl
uo976@connect.hkust-gz.edu.cn

Qiang Wang
哈尔滨工业大学（深圳） 中国
深圳
qiang.wang@hit.edu.cn

Wei Wang
香港科技大学 中国香港特别行政区
weiwa@cse.ust.hk

Xiaowen Chu
香港科技大学（广州） 中国广州 香
港科技大学 中国香港特别行政区
xwchu@ust.hk

arXiv:2603.17435v1 [cs.DC] 18 Mar 2026

摘要

无损模型压缩在缓解位精确大语言模型（LLM）服务中的内存和带宽瓶颈方面前景广阔。然而，现有方法往往会导致推理速度显著下降，因为其设计与 GPU 架构存在根本性不匹配：在内核层，传统熵编码器生成的变长比特流会破坏 SIMT 并行性；在系统层，解耦流水线会造成冗余内存流量。我们提出 ZipServ，这是一种为高效 LLM 推理协同设计的无损压缩框架。ZipServ 引入了张量核心感知三重位图编码 （TCA-TBE），这是一种新型定长格式，可实现恒定时间并行解码；同时还配备了融合解压缩与通用矩阵乘法 （ZipGEMM）内核，可将权重直接按需解压至张量核心寄存器中。这种“加载压缩数据，计算解压数据”的设计消除了中间缓冲区，并最大限度地提高了计算强度。实验表明，ZipServ 可将模型大小最多缩减 30%，相较于英伟达的 cuBLAS 实现最高 2.21× 倍的内核层加速，并可加速端到端推理平均比 vLLM 高 1.22× 。ZipServ 是首个同时实现存储节省并显著加速图形处理器上大语言模型推理的无损压缩系统。

CCS 概念： •**计算方法** → **共享内存算法**。

关键词： 大语言模型推理、无损压缩、通用矩阵乘法、图形处理器、张量核心

ACM 引用格式：
Ruibo Fan、Xiangrui Yu、Xinglin Pan、Zeyu Li、Weile Luo、Qiang Wang、Wei Wang 和 Xiaowen Chu。2026 年。ZipServ：借助硬件感知无损压缩实现快速且内存高效的大语言模型推理。载于 第 *31* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *2* 卷（*ASPLOS '26*），*2026*年*3*月*22*日至*26*日，美国宾夕法尼亚州匹兹堡。 ACM，纽约，纽约州，美国，17 页。
https://doi.org/10.1145/3779212.3790250

本作品根据知识共享署名 4.0 国际许可协议授权。

ASPLOS '26，美国宾夕法尼亚州匹兹堡©2026 版权所有归所有者/作者所有。ACM ISBN
979-8-4007-2359-9/2026/03https://doi.org/10.1145/3779212.3790250

1 引言

GPT 等大语言模型（LLM）的变革性力量4 [54], LLa MA3 [17], 和通义千问3 [70]源于其庞大的规模 [3, 36], ，这使全新的人工智能应用范式成为可能 [6, 60, 74, 81]。然而，如此庞大的规模也带来了显著的部署挑战，使 GPU内存和内存带宽成为大语言模型服务的主要瓶颈，在资源受限的环境中尤为如此。

模型压缩为高效部署 LLM 提供了一种很有前景的解决方案。现有的大多数方法都是有损的，通过以下方式对模型权重进行近似以减小规模：

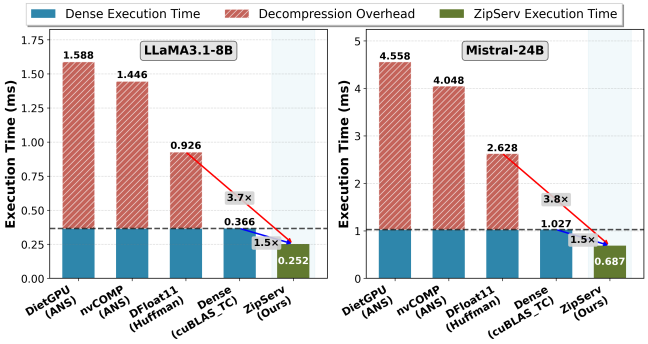

图 1. NVIDIA L40S GPU 上 GateUp_Proj 层的无损压缩流水线执行时间。

量化（例如 GPTQ [23], AWQ [43]）或剪枝（例如 SparseGPT [22]）。然而，此类近似存在精度损失的风险。例如，激进的 4 位量化（例如 MXFP4）会使 LiveCodeBench 上的准确率从 56.0% 降至 36.2% [44],，即使是稳健的 int8 量化（GPTQ- int8），也可能导致长上下文推理（NOCHA）最高 11.1% 的损失 [49]。这些风险削弱了其在安全关键型和面向用户的场景中的可靠性，因而促使人们探索能够保证位精确可复现性和数值完整性的方法。

无损压缩能够以位精确的方式表示模型，且不会造成精度损失，因此是一种颇具吸引力的替代方案。迄今为止，其优势主要体现在存储和训练流程中。例如，LMC [71] 和 ZipNN [29] 采用 Huffman [31] 压缩模型检查点，以实现高效存储和分发；NeuZip [28] 和 DietGPU [33] 则减少训练期间的内存和通信开销。尽管近期的一些工作，尤其是 DFloat11 [85],，试图将这些优势扩展到推理，但实际效率仍难以实现。将现有无损技术集成到服务流水线中时，会产生显著的运行时间开销。如图 1所示，仅解耦的解压缩步骤就需要核心推理计算44× 时间的 1.56–3 倍。这种开销迫使人们在内存效率和运行时间效率之间作出令人遗憾的取舍。

我们认为，这种权衡并非根本性的，而是源于传统压缩算法与现代 GPU 架构之间的不匹配。这个问题体现在两个层面。在**内核层**，传统熵编码器（例如 Huffman [31] 或 ANS [18]）会生成变长比特流，其解码需要串行执行依赖于数据的操作。这些操作并不适合 GPU 线程束步调一致的并行 SIMT 执行模型，导致严重的控制流分歧和计算资源利用不足。在**系统层面**，大多数框架采用解耦式推理流水线：权重在被内核使用前，会先完整解压缩到全局内存缓冲区中。这种分阶段执行会造成冗余且延迟较高的内存访问，抵消压缩带来的带宽节省，并降低推理过程中的算术强度。

为解决这些根本性的算法与硬件不匹配问题，我们提出**ZipServ**[1],，这是首个专为 GPU 上高性能大语言模型推理而协同设计的无损压缩框架。我们观察到，当代模型中的大语言模型 BFloat16 权重，其指数位分布高度偏斜、熵较低。利用这种统计冗余，我们提出张量核心感知三重位图编码（TCA-TBE），一种面向 GPU 架构定制的定长位图权重格式。与变长熵编码器不同，TCA-TBE 通过轻量级位运算实现恒定时间并行解码，从而消除控制流分歧，并与 GPU 的 SIMT 执行模型相契合。基于 TCA-TBE，ZipServ 设计了融合解压缩-*GEMM*内核（ZipGEMM）。ZipGEMM 不会将权重解压缩到全局内存作为中间步骤，而是即时解码，将压缩权重直接送入为张量核心矩阵乘法单元供数的寄存器文件中。这种"加载压缩数据，计算时解压缩"设计消除了中间缓冲区，减少了数据移动，并最大限度地提高了计算与内存访问之间的重叠。通过共同解决内核层和系统层的不匹配问题，ZipServ 将无损压缩在理论上的存储节省转化为推理优化型 GPU 上切实的性能提升。

我们通过与最先进的无损压缩方法进行全面基准测试，展示了 ZipServ 的有效性，包括 DietGPU [33],供应商优化的 nvCOMP [53], 以及基于 Huffman 的 DFloat11 [85]。与这些普遍存在显著运行时间开销的基线方法相比，ZipServ 在多种推理优化型 GPU 上，无论在内核层还是系统层，都始终实现了显著加速，其中包括 RTX4090、L40S 和 RTX5090。我们的融合 ZipGEMM 相较于英伟达的 cuBLAS，最高可实现 2.21× 倍加速；相较于 DFloat11（最快的无损压缩流水线），最高可实现 5.53× 倍加速。这些内核级改进转化为端到端平均 1.22×倍加速，优于 vLLM [39]等领先系统。我们的结果首次表明，与硬件协同设计的无损压缩既能节省存储空间，也能显著加速大语言模型推理。

本文的主要贡献如下：

- 我们揭示了传统基于熵的压缩与图形处理器架构之间的根本不匹配，并发现了阻碍高效推理的内核级和系统级瓶颈。
- 我们提出了 TCA-TBE，这是一种专为 SIMT执行和张量核心分块设计的定长、基于位图的编码，可实现常数时间并行解码。
- 我们设计了 ZipGEMM，这是一种新型内核，可直接在张量核心

[1]公开地址：https://github.com/HPMLL/ZipServ_ASPLOS26.git

寄存器中即时执行解压缩，消除中间内存缓冲区并最大化计算强度。

- 我们提出并评估了 ZipServ，这是一种无损压缩的大语言模型推理框架，可在各种大语言模型和 GPU 上实现端到端加速，首次提供了无损压缩能够直接加速大语言模型服务的实用证据。

2 背景

2.1 基于 Transformer 的大语言模型

基于 Transformer 的大语言模型 [2, 17, 70] 由多头注意力、前馈网络（FFNs）和归一化层堆叠而成。在推理过程中，计算以自回归方式分为两个阶段进行：预填充和解码。预填充阶段会对输入提示上的计算进行并行处理，由于在多个词元上执行大规模矩阵乘法，因此具有较高的算术强度。相反，解码阶段每次生成一个词元，此时矩阵乘法涉及每个批次元素中的单个词元。因此，解码阶段的计算利用率较低，并且对内存带宽更加敏感。在这两个阶段中，占主导地位的操作都是稠密矩阵乘法： $Y = WX$，其中 $W \in \mathbb{R}^{M\times K}$是经过学习的权重矩阵， $X \in \mathbb{R}^{K\times N}$ 是激活值，其中 M 是输出维度， K 是隐藏维度， N 是词元数量。

2.2 BFloat16 格式

BFloat16（BF16） [35] 是一种 16 位浮点格式，已成为大语言模型推理的事实标准精度格式，在内存效率与数值稳健性之间实现了平衡。它原生支持主要的硬件加速器，包括 NVIDIA Tensor Cores [47], Google TPU [34], 和 Intel AMX [37]，并广泛应用于生产级模型，包括 LLaMA-3 [17], 通义千问 [70], 和 Mistral [2]。一个 BF16数由 1 个符号位、8 个指数位和 7 个尾数位组成。其数值计算方式如下：

$$\mathrm{BF16}(x) = (-1)^{\mathrm{sign}} \times 2^{\mathrm{exponent}-127} \times (1.\mathrm{mantissa}).$$

这种布局保留了 IEEE FP32（1-8-23）的完整指数范围，同时降低了尾数精度。与 FP16（1-5-10）相比，BF16 具有更宽的动态范围，从而降低了大型模型发生上溢和下溢的风险。

2.3 GPU 架构与张量核心执行

现代 GPU 由多个流式多处理器（SMs）构成，每个流式多处理器均配备 SIMT 核心、Tensor Core、寄存器、共享内存和本地缓存。线程被分组为每组 32 个的线程束，并遵循单指令多线程（SIMT）范式执行。Tensor Core 是专用于高吞吐量矩阵乘法的处理器。在较新的 NVIDIA 架构上， [50, 51], Tensor Core 通过 PTX 级别的

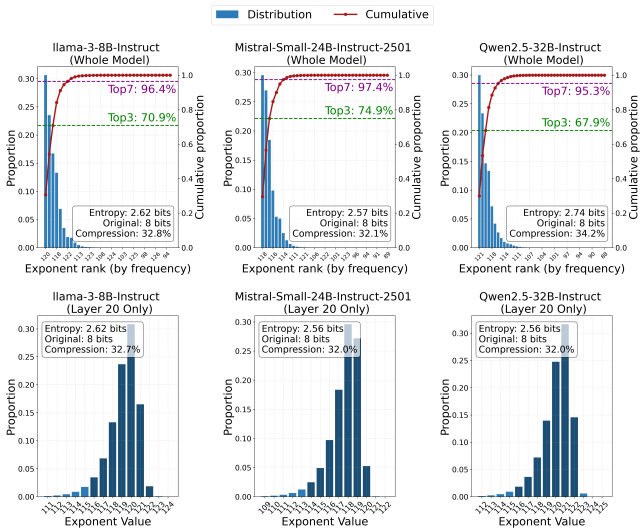

图 2. 大语言模型权重中的指数位分布。

mma.sync.m16n8k16 指令支持 BF16 操作数，该指令可在小型矩阵图块之间执行融合矩阵乘加（FMA）操作。典型的 BF16 Tensor Core 操作可表示为：

$D_{\mathrm{frag}} = A_{\mathrm{frag}} \times B_{\mathrm{frag}} + C_{\mathrm{frag}}$，其中 $A_{\mathrm{frag}} \in \mathbb{R}^{16\times16}$、$B_{\mathrm{frag}} \in \mathbb{R}^{16\times8}$ 和 $C_{\mathrm{frag}} \in \mathbb{R}^{16\times8}$ 是 FP32累加器片段。该操作在线程束级别执行，由一组 32 个线程协同完成矩阵乘法计算。输入和输出片段分布在整个线程束中。每个线程在其寄存器中保存片段元素的特定子集，完整片段则由所有线程共同形成。

3 差距与机遇

无损压缩可实现位精确的模型表示，但由于传统编解码器与 GPU 架构不匹配，导致运行时间开销高，因此很少用于推理。本节量化了大语言模型权重的可压缩性，并指出了促使我们采用协同设计解决方案的关键内核和系统级瓶颈。

3.1 BF16 权重的可压缩性

我们分析了领先大语言模型的 BF16 权重，包括 Llama-3-8B-Instruct [17],Mistral-Small-24B-Instruct-2501 [2], 和 Qwen2.5-32B-Instruct [69], , 并观察到其 8 位指数域中存在显著冗余。如图 2所示，指数分布 高度偏斜：出现频率最高的 **前 3 个** 指数占全部权重的 67% 以上，而 **前 7 个** 指数覆盖超过 **95%**（例如，Llama-3 中为 96.4 %，Mistral-24B 中为 97.4%）。指数域的信息熵仅为 **2.57–2.74 比特**，远低于其 8 位的分配，意味着理论上的无损压缩比约为 1。51×（16/10.6），适用于 BF16 值。这些发现与此前的研究

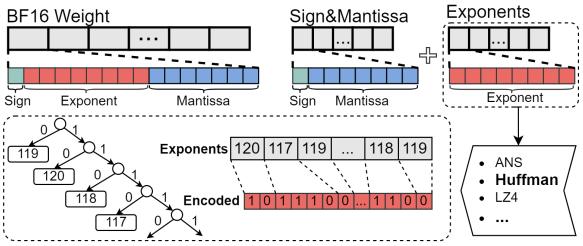

图 3. BF16 权重的现有无损压缩方法。以霍夫曼编码为例。

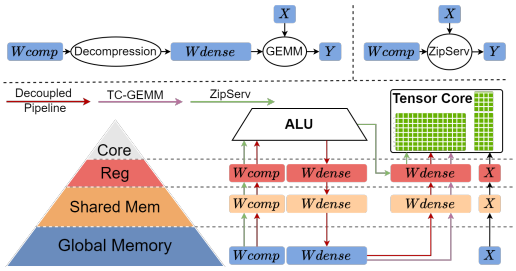

图 4. 现有的无损压缩推理流水线。

结果一致 [28, 29, 71, 83, 85]。我们进一步考察了四个大语言模型系列（Gemma-3、Mistral、通义千问2.5 和 LLaMA-3.1）中 3,875 个权重矩阵的这种冗余，揭示出一个关键结构特性：指数连续性。在其中 99.6% 的矩阵中，出现频率最高的前 7 个指数构成数值上连续的序列（即 $e^{\star}$、…、$e^{\star}+6$）。因此，一个简单的连续窗口平均可覆盖全部权重的 97.1%，接近信息论极限。在附录A中，我们证明这并非巧合，而是大语言模型的内在属性。这种连续性是ZipServ的基石。它无需使用复杂且不利于硬件实现的变长编解码器（例如 Huffman），而采用 **定长**的基数加偏移表示。这一洞见直接促成了我们的张量核心感知三重位图编码（TCA-TBE）及其隐式查找机制，详见第 §4.3.2节。

3.2 内核级架构不匹配

现有方法通过对指数流进行熵编码，利用 BF16 权重中的指数冗余。例如，DFloat11 使用霍夫曼编码 [85],，而 Diet-GPU 使用非对称数制系统（ANS） [33]。如图 3所示，这些方法会生成由变长符号组成的压缩比特流，符号长度取决于其统计频率。然而，必须按顺序对该比特流进行解压缩，才能正确恢复每个指数；这从根本上与现代 GPU 锁步、大规模并行的 SIMT 执行模型相冲突。

为了说明这种失配，我们考察 DFloat11 [85] 中的三阶段解压缩流水线。❶ **比特流划分**。比特流被拆分为数据块，以便线程并行处理。然而，由于变长符号会跨越数据块边界，线程无法独立运行，而是需要额外的元数据来定位有效符号的起始位置，这会带来开销并干扰并行执行。❷ **符号提取**。线程使用分层查找表（LUTs）进行符号解码——这是一种依赖数据的操作。当线程束线程遇到不同长度的符号时，较快的线程会等待较慢的线程，导致分歧并降低 GPU 资源利用率。❸ **指针推进**。符号解码完成后，每个线程都会将其比特指针前移相应的符号长度，而该长度只有在查找完成后才知道

。这使解码循环不可避免地串行化，并牺牲了利用指令级并行的机会。我们的评估显示，在 L40S GPU 上，即使是高度优化的解压器（例如基于 ANS 的 DietGPU 和基于哈夫曼编码的 DFloat11），其达到的内存带宽也分别仅为峰值的 43.7% 和 76.5%。这种低效揭示了算法与硬件之间的根本失配：熵编码本质上依赖数据，而高效的 GPU 执行则需要规则、统一的并行性。

3.3 解耦推理流水线的低效性

熵编码解码在架构上的低效性不仅体现在内核层，也体现在大语言模型推理的系统流水线层面。在主流方法中，解压缩作为一个独立、解耦的预处理阶段执行（见图4）：它先将全部解压后的权重写入全局内存，然后再将其传递给计算内核。这种解耦流水线设计会导致冗余数据传输，削弱压缩的优势，尤其是在带宽受限的环境中。我们使用Roofline模型，通过聚焦计算强度（CI）对其低效性进行定量分析。

计算强度。CI 衡量每从全局内存读取一个字节所执行的浮点运算次数。对于典型的 BF16通用矩阵乘法运算 $Y_{M\times N} = W_{M\times K}X_{K\times N}$，计算强度为：

$$CI_{GEMM} = \frac{MNK}{MK + KN + MN}. \tag{1}$$

在解耦流水线场景中，假设平均压缩比（CR）为 1.51（§3.1），则 CI 为：

$$CI_{\text{Decoupled}} = \frac{2MNK}{MK\left(\frac{2}{\text{CR}} + 4\right) + 2(KN + MN)} \approx \frac{MNK}{2.66MK + KN + MN}. \tag{2}$$

Roofline模型分析。图5 展示了英伟达 RTX4090 上的屋顶线分析。在解码阶段，标准GEMM和解耦流水线都处于内存受限区间，性能随计算强度线性增长。然而，我们的分析凸显了解耦方案的显著代价：为生成中间解压权重而产生的额外内存流量大幅降低了计算强度。具体而言，对于一个

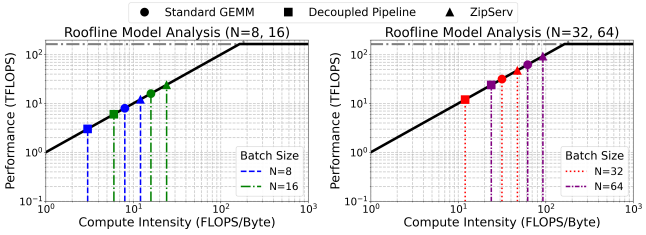

图 5. 屋顶线分析。

大小为 $M = K = 4096$ 的权重矩阵，批量大小为 8、16、32 和 64 时，解耦流水线相较于标准GEMM的计算强度分别下降了 62.3%、62.2%、62.0% 和 61.7%。

ZipServ的融合设计。 解耦流水线效率低下，直接源于将解压缩后的权重暂存在全局内存中。ZipServ 通过引入融合解压缩-GEMM内核来解决这一问题：该内核直接从动态随机存取存储器中读取压缩权重，并在运行时将其解压缩至寄存器文件，随后立即供张量核心使用。这种方法有效提升了 CI 至

$$CI_{\text{ZipServ}} = \frac{2MNK}{MK \cdot \frac{2}{\text{CR}} + 2(KN + MN)} \approx \frac{MNK}{0.66MK + KN + MN}. \quad (3)$$

重新审视图5中的屋顶线模型，ZipServ的融合执行（CI_{ZipServ}）展现出显著提升，其 CI 甚至比未压缩的 GEMM 基线高出约 50%。这一优势在内存受限区间尤为明显，可带来相对于压缩比的线性加速，将信息论冗余转化为实际运行时间加速。

4 ZipServ 的设计

我们先前的分析指出了阻碍大语言模型推理中无损压缩解码的两类低效来源：内核级和系统级。本节介绍 ZipServ，这是一种针对存储效率与快速、位精确的大语言模型推理协同设计的无损压缩系统。

4.1 概述与工作流程

如图6所示，ZipServ由两个主要组件构成：一个离线压缩器，用于将 BF16 模型权重转换为并行化友好的压缩表示；以及一个在线推理引擎，负责在运行时高效解码和计算。

离线压缩器。离线压缩器的核心是张量核心感知三重位图编码(TCA-TBE)，这是一种定长、基于位图的压缩格式，旨在支持并行解码，通过 GPU SIMT 执行和张量核心加速的 GEMM 运算实现。如**算法1**所述，给定一个模型，压缩器会先分析每一层权重的指数分布。它不会任意选择高频指数，而是识别一个 k 数值连续的指数值窗口（通常为$k = 7$），使其能够最大程度覆盖权重分布。压缩器将该范围前一个值记录为基准指数（BaseExp，即 min(range) − 1）。利用这一范围，压缩器将整个权重矩阵编码为 TCA-TBE 表示形式。每个 8×8 权重图块都会转换为三个 64 位位图和两个紧凑的值缓冲区：一个用于存储落在所选指数范围内的高频值（仅存储相对于基准指数的符号位和尾数），另一个则以完整的 BF16 精度存储离群值。随后，压缩后的模型会加载到图形处理器上，准备提供服务。

在线推理引擎。该推理引擎采用阶段感知策略，针对预填充阶段和解码阶段调整执行流水线，同时统一使用 TCA-TBE 格式。在受计算限制的**预填充阶段**，引擎采用解耦执行：专用解压缩内核先将权重解压缩到全局内存，随后执行预填充计算。这种方法使高吞吐量的通用矩阵乘法能够有效摊销解压缩开销。在受内存限制的**解码阶段**，引擎则切换至融合解压缩-GEMM内核（ZipGEMM）。ZipGEMM 支持“加载压缩数据、计算解压缩数据”的执行模式，将权重直接按需解压缩至张量核心寄存器。这消除了冗余数据传输，并最大限度地提高每个令牌生成过程的计算强度。这两种专用执行路径可实现近乎最优的推理性能。

4.2 张量核心感知三重位图编码

ZipServ 构建于一种新颖的张量核心感知三重位图编码（TCA-TBE）方案之上。该方案旨在最小化权重内存占用，同时支持在 GPU 上进行高效的并行解码。与现有基于可变长度比特流的熵编码器不同，TCA-TBE 采用定长分块结构表示，从而确保恒定时间的线程本地解压缩。其数据布局经过精心设计，与张量核心分块和寄存器级操作数分布相匹配，使解压缩后的权重能够直接由 mma.sync 指令使用。TCA-TBE 的核心是为每个权重元素分配一个固定长度的 **3 位码字**，用于表示八种可能的状态之一（000–111）。在离线压缩过程中，ZipServ 对权重矩阵的指数直方图进行分析，找出出现频率最高的 7 个指数值，并将其映射到码字 001–111。特殊码字 000 用作回退标记，表示指数不属于前 7 个指数值的权重；这些权重随后以全精度存储。

码字长度的选择。我们选择 3 位码字，因为它利用了当代大语言模型中观察到的高度偏斜的指数分布，实现了接近最优的压缩比。为了量化这一设计

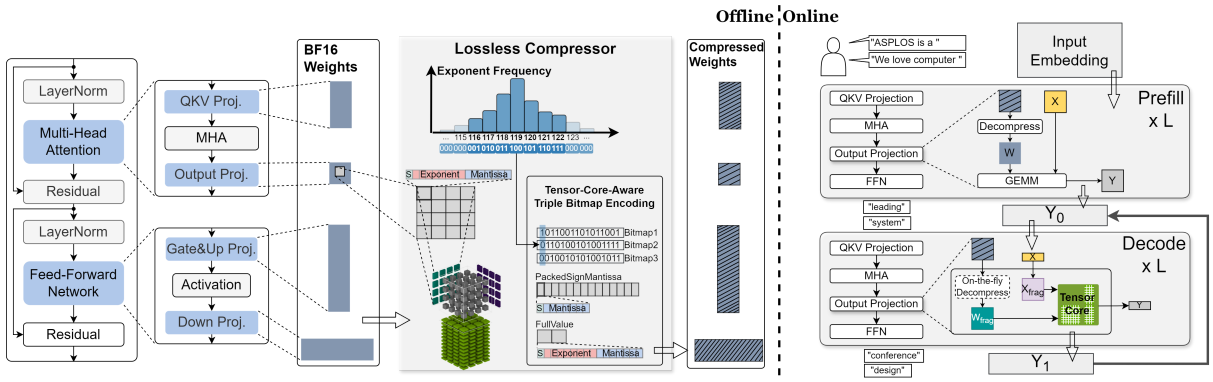

图 6. ZipServ 概览。ZipServ 包含一个离线无损压缩器（左）和一个在线推理引擎（右）。

算法 1 ZipServ 离线压缩器（TCA-TBE）

输入： 权重矩阵 W，分块大小 $T = 8 \times 8$
输出： 位图 $B_{1..3}$、高频缓冲区 H、回退缓冲区 L，基准指数 e_{base}

```
▷ 阶段 I：全局指数分析
Hist ← 计算指数直方图(W)
E_top ← 选择排名前 7 的连续指数(Hist)
e_base ← 最小值(E_top) − 1   ▷ 设置隐式查找的基准
▷ 第二阶段：瓦片编码
对于每个图块 t ∈ W 执行
    初始化局部位图 b1, b2, b3 ← 0
    对于 i = 0 到 63 执行
        w ← t[i];  e ← w.exponent
        如果 e ∈ E_top 则
            c ← e − e_base   ▷ 计算 3位编码 c ∈ [1, 7]
            b1[i] ← c0;  b2[i] ← c1;  b3[i] ← c2   ▷ 设置位
            H。推入(打包(w.sign, w.mantissa))
        else
            L。Push(w)   ▷ 存储全精度回退
        结束 if
    结束 for
    将 b1, b2, b3 存储到全局 B1..3
结束 for
```

选择，我们计算每元素存储成本的期望值如下：

$$AverageBits(n) = r_n \cdot (n + 8) + (1 - r_n) \cdot (n + 16),$$

其中 n 表示码字长度，r_n 表示前 $2^n - 1$ 个指数值所覆盖的权重比例。如§3.1所示，$r_3 \approx 0.96$，平均每个元素占用 11.3 位，接近理论下界（8+2.6=10.6 位），明显优于 2 位（12.4 位）和 4 位（12.1 位）码字。此外，3位编码生成紧凑的 7 项码本，可通过简单查表进行解码。每个线程只需执行少量位运算，便可利用 warp同步张量核心流水线高效完成。

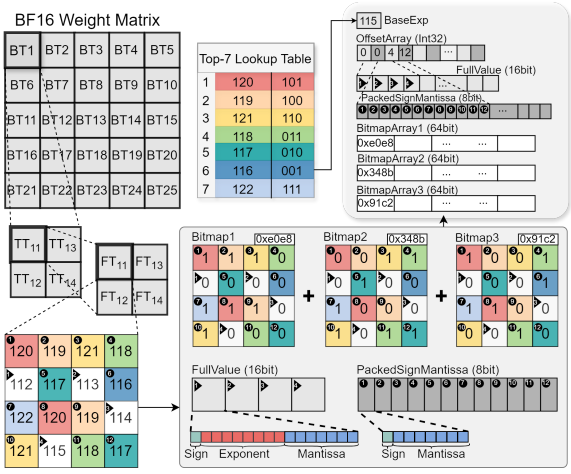

图7。张量核心感知三重位图编码。此处展示的4×4 FragTile仅用于说明；实际大小为 8×8。

解耦三重位图布局。为最大限度提高 SIMT架构上的解码效率，TCA-TBE采用解耦三重位图布局，而非将码字打包成密集位流。传统位流在 GPU 上效率低下，因为打包非字节对齐的编码（如 3 位编码）会迫使码字跨越内存字边界。这需要复杂的非对齐访问逻辑，还会引入数据依赖分支，进而导致线程分歧，严重降低 SIMT 吞吐量。

TCA-TBE 将每个 8×8 权重块的 3 位码字分解为三个相互独立的 64位位图，每个位图表示一个位平面（图 7）。这一设计带来两项优势。首先，由于每个位图都是连续的 64 位字，且自然对齐到原生内存边界，因此可以保证合并内存访问。其次，它支持无分支解码。线程束中的所有线程都遵循相同的执行路径，这与现代 GPU 上的 SIMT模型相契合。

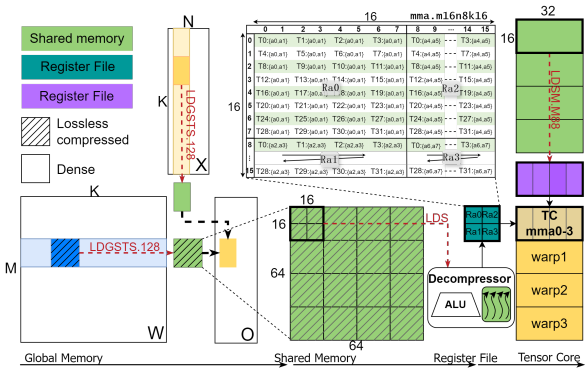

图 8. 数据移动与指令流水线。

分层分块设计。TCA-TBE 采用三层分层分块方案，按照现代 GPU 的架构粒度对权重矩阵进行划分。❶ FragTile（FT）：基本单元是一个 8×8 图块，与 Tensor Core 指令中最小的操作数片段相匹配。❷ TensorCoreTile（TT）：每个 16×16 图块由一个 2×2 FragTile 网格组成。其大小与 PTX 级 Tensor Core mma 指令（mma.m16n8k16）所需的操作数维度（m=16, k=16）相符。❸ BlockTile（BT）：在最粗粒度层级，一个 64×64 图块汇集多个 TensorCoreTile，并由一个线程块协同处理。TensorCoreTile 中的 FragTile 按列优先顺序存储，与 Tensor Core 指令所需的操作数寄存器布局（例如 Ra0–Ra3）保持一致。这一设计消除了运行时坐标变换的需要，降低了指令开销。每个 8×8 FragTile 使用五个缓冲区进行编码。❶ 三个位图，每个位图表示 3 位码字的一个位平面。❷ 一个 PackedSignMantissa 缓冲区，用于存储指数属于最高频类别的权重的紧凑 8 位表示（符号位和尾数）。❸ 一个 FullValue 缓冲区，用于存储未被指数码本涵盖的权重的全精度 BF16 值。在矩阵层级，TCA-TBE 将这些缓冲区组织为四个连续的全局数组，每个数组均按照分块层级嵌套。此外，偏移数组记录了 PackedSignMantissa 和 FullValue 数组中每个 GroupTile 的起始偏移量。

4.3 融合 ZipGEMM 内核设计

TCA-TBE 面向 SIMT 的设计为高吞吐量解码带来了新的机遇。为此，ZipServ将解压缩与矩阵乘法融合到一个名为 **ZipGEMM** 的内核中。该内核以紧凑的 TCA-TBE 格式从全局内存中读取权重，并在计算过程中即时解压缩。ZipGEMM 支持加载压缩数据、计算时解压缩 的执行模式，大幅降低了解码阶段每次令牌生成所需的内存带宽（见图 5）。

4.3.1 内核工作流程。图8 展示了 ZipGEMM 内核的工作流程。该内核采用 split-K 分块架构，每个线程块以数据块为单位迭代处理 K 维度。每次迭代中，内核依次执行四个协同阶段。❶ 分块加载。线程协同地将压缩权重块和对应的激活块从全局内存加载到共享内存，并使用异步、向量化的内存指令（即LDGSTS.128）绕过 L1 缓存，从而提升全局内存带宽利用率。每个图块中的PackedSignMantissa和FullValue 数组均在离线阶段进行填充，以确保 128 位对齐。❷ Warp 级解码。每个线程束独立地从共享内存中解压缩压缩权重。解压器使用轻量级 ALU 操作重建原始 BF16 值，使其布局适用于张量核心，同时避免共享内存往返访问。❸ 激活值寄存器传输。使用 LDSM.M88 指令将激活块从共享内存移入寄存器。该指令使线程束能够加载一个 16×16 图块，并将其排列成张量核心所需的布局。❹ 张量核心计算。当解压缩后的权重和激活值都存入寄存器后，线程束执行张量核心 mma 指令。该执行路径与标准 cuBLAS GEMM 内核非常相似，同时可直接处理压缩表示，并减少全局内存访问。

4.3.2 高效解压器。 ZipGEMM 集成了一种高效解压器，可在寄存器文件中直接进行线程局部的压缩权重重建。该解压器的核心原理是 每个线程独立解压正确张量核心片段布局所需的元素。具体而言，如图 7所示，片段布局要求线程 i的.bf16x2 寄存器（例如 Ra0）保存位于位置$2i$ 和 $2i + 1$ 的值，这些位置位于 8×8 图块中，分别记为 a_0 和 a_1。由于每个元素采用两种状态之一进行编码——高频定长编码或回退全精度值——且这些状态以非结构化方式分布，因此解压器要解决的是一个稀疏、非均匀的空间重建问题。在此情境下会遇到两个挑战。首先，每个线程必须高效确定其所分配元素的状态（压缩或回退）。其次，每个线程应以确定性且适合 SIMT 的方式恢复原始 BF16 表示。为此，ZipServ的解压器由三个紧密集成的阶段构成：空间位图指示器、动态寻址和快速指数重组（见图9 和算法 2）。

空间位图指示器。 每个线程首先通过评估空间指示器掩码，确定其所分配元素的存储模式。在离线压缩期间，每个 8×8权重块使用三个 64 位位图进行编码，其中每个位图编码 3 位码字中的一个比特。

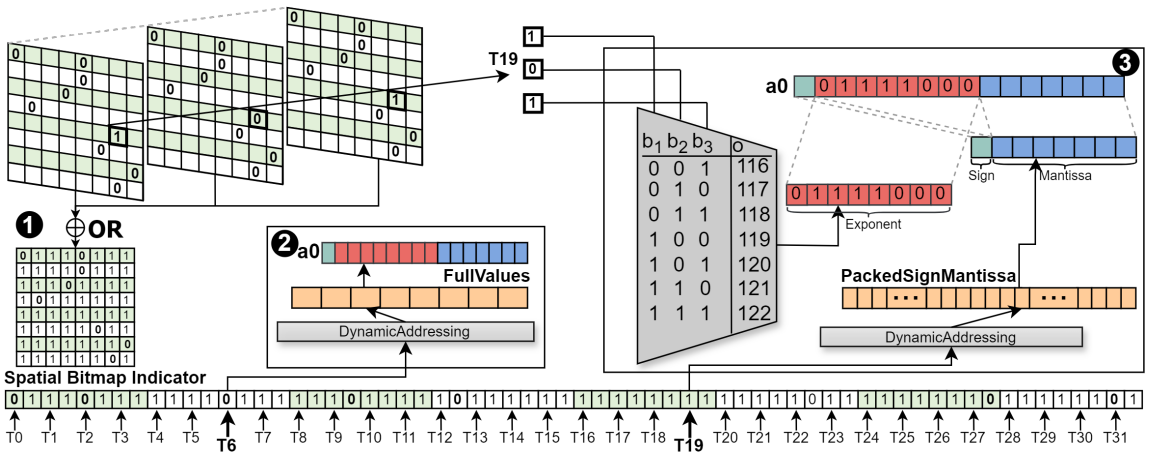

图 9。解压器设计。

算法 2 ZipGEMM 线程本地解压

输入： 位图 $B_{1..3}$，缓冲区 H，L，基准指数 e_{base}，通道标识符 l
输出： 寄存器对 R，其中包含两个 BF16 值

▷ 步骤 1：构建空间指示器
$M \leftarrow B_1 \vee B_2 \vee B_3$
▷ 步骤 2：并行解压缩元素
对于 $k \in \{0,1\}$ **执行循环**
  $p \leftarrow 2 \cdot l + k$ ▷ 8×8 图块中的全局位置
  $mask \leftarrow (1 \ll p) - 1$
  $idx_{\mathcal{H}} \leftarrow \mathrm{POPC}(M \,\&\, mask)$ ▷ 计算索引
  如果 $(\mathcal{M} \gg p)\ \&1$ **则**
    ▷ 案例 A：高频路径
    $val \leftarrow \mathcal{H}[开始_{\mathcal{H}} + idx_{\mathcal{H}}]$ ▷ 获取符号位 + 尾数
    ▷重构3位编码
    $c \leftarrow (B_3[p] \ll 2) \vee (B_2[p] \ll 1) \vee B_1[p]$
    $e \leftarrow e_{base} + c$ ▷ 隐式查找
    $w_k \leftarrow \mathrm{MakeBF16}(val.sign,\ e,\ val.mantissa)$
  else
    ▷ 情况B：回退路径
    $idx_{\mathcal{L}} \leftarrow p - idx_{\mathcal{H}}$ ▷ 计算回退路径中的索引
    $w_k \leftarrow L\ [开始_{\mathcal{L}} + idx_{\mathcal{L}}]$
  结束 if
结束 for
$R \leftarrow$ 打包寄存器(w_0, w_1)
返回 R

位图编码 3 位码字中的一个比特。在运行时，三个位图通过线程束级按位或合并，生成一个 64 位指示掩码。该掩码中的每一位指定一个元素的存储模式：1 表示压缩（高频），0 表示回退（未压缩）。每个线程通过检查空间指示器掩码中对应的位来确定其解码路径，该掩码存储在寄存器中。具体而言，位置 $2i$ 处的位（对于 a_0）和位置 $2i+1$ 处的位（对于 a_1）指示状态这两个已分配元素之一。例如，线程 19 发现位 38 (2×19) 已置位，这表示其 a_0 元素以压缩形式存储。它从高频缓冲区中获取打包值，并继续进行指数重组。相比之下，线程 6 发现位 12 (2×6) 未置位，因此直接从回退缓冲区加载其 a_0。这一按位决策过程开销很小，完全驻留于寄存器中，并且能在常数时间内完成。

动态寻址。 确定存储模式后，每个线程都会即时计算其在相应值缓冲区中的读取偏移量，无需显式的逐元素索引。这通过对空间指示器执行轻量级的线程束内前缀和来实现。对于线程 i，偏移量的计算方法是：统计空间指示器的位 $[0, 2i-1]$ 中，之前出现的同一存储类型元素的数量。具体而言，如果元素是压缩的（位 = 1），偏移量等于 1 的数量；如果是未压缩的（位 = 0），则等于该范围内 0 的数量。借助 __popc() 和 __shfl_sync() 等 GPU 原生指令，可以高效地完成这些计数。例如，线程 6 遇到位置 12 处未置位的位时，会通过统计位 [0, 11]中的 0 的数量来计算其回退缓冲区偏移量。线程 19 遇到置位的位 38 时，则统计位 [0, 37] 中的 1 的数量，以访问压缩缓冲区。这种动态寻址机制将索引操作转化为确定性的、适合 SIMT 的操作，并且与 GPU 执行模式自然契合。

通过隐式查找快速重组指数。 为进一步降低解码开销，ZipServ 使用基于算术重映射的隐式查找机制重建指数，从而避免基于查找表的解码。在离线压缩期间，系统会在全局范围内找出出现频率最高的 7 个指数值，并为其分配 3 位码字（001–111），按数值递增顺序排列，而非按频率排名排列。系统会记录一个全局基准指数 base_exp = min(top_exponents) − 1，供所有

图块共享。在运行时，每个线程通过将 3 位码字加到基准指数上，重建原始指数。此操作只需一条整数算术逻辑单元指令，即可避免访问共享内存中的查找表。随后，将恢复出的指数与符号位和尾数字段合并，组成有效的 BF16 值。例如，线程 19 发现空间指示器中的第 38 位已置位，并通过读取三个位图平面中对应的位来重建 3 位码字，得到 101（5）。当全局基准指数为 115 时，它恢复出原始指数 $115 + 5 = 120$，再将其与符号位和尾数合并，生成最终的 BF16 值。这种算术解码过程完全兼容 SIMT，并充分利用图形处理器的整数流水线。

重新打包为 Tensor Core 片段。每个线程将重构出的两个 BF16 元素重新打包到一个 bfloat162 寄存器中，以匹配 Tensor Core mma.sync 指令所需的操作数布局。

4.3.3 细粒度软件流水线。ZipGEMM 使用分层的双级流水线，使内存传输、解压缩和计算相互重叠，从而有效隐藏内存和解压缩延迟。在粗粒度层面，按图块进行的双缓冲使全局内存到共享内存的传输与计算重叠；在细粒度层面，按切片交错处理，使共享内存到寄存器的数据移动和解压缩与 Tensor Core 操作重叠。该机制通过两个共享内存缓冲区实现，用于存储压缩权重（三重位图、打包的符号位与尾数、回退值）和激活值。在每个图块内，计算沿 K 维度切分（通常为 16×16 个片段），并采用交错式加载-解压缩-计算模式进行处理。当 Tensor Core 对切片 i 执行矩阵乘法（mma）时，算术逻辑单元会并行地从共享内存加载并解压缩切片 $i+1$ 的权重到寄存器中。这样便能在计算过程中隐藏解压缩和内存延迟，确保计算持续进行。

为协调两个流水线层级，ZipGEMM采用分层屏障策略，实现瓦片间和线程束内同步。**瓦片间同步：** cp.async.wait_group<0>() 和__syncthreads() 确保所有异步传输在切换缓冲区之前完成。该屏障设置在最后一个切片解压之后、最终切片 mma 之前，使计算能够在加载和解压下一个瓦片的同时继续进行，从而最大限度地提高重叠度并减少停顿。**线程束内协调：** 线程束内操作通过 SIMT 模型隐式同步，因此在切片级别的加载、解压和计算之间无需显式屏障。

4.4 阶段感知推理策略

ZipServ 仅在解码阶段使用融合 ZipGEMM 内核，以加速令牌生成。对于计算受限的预填充阶段，由于较大的矩阵维度（$N = BS \times Seq_len$）具有较高的算术强度，

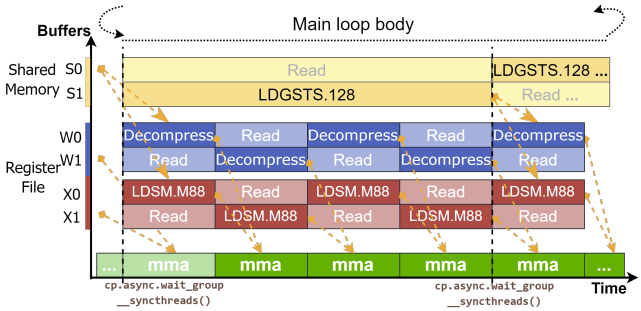

图10。分层软件流水线设计。

ZipServ会切换到解耦流水线：高效的解压缩内核首先将压缩权重提取到全局内存中，然后执行高吞吐量的 GEMM 运算，以摊销解压缩开销（通常为 <4%，如 §6.4 所示）。在预填充和解码阶段，解压缩内核与 ZipGEMM 内核均采用相同的压缩格式和逐线程解压缩逻辑（§4.3.2），因此无需进行运行时格式转换。

5 实现

我们将 ZipServ 实现为高性能、模块化的推理后端，代码量约为 3.5K 行。核心引擎约由 2.5K 行 CUDA 和 C++ 代码组成，实现了离线 TCA-TBE 压缩器和在线 ZipGEMM 内核。该内核使用 nvcc 编译为独立的共享库（.so），并通过 C++ API 提供权重打包和内核启动功能。其余 1.0K 行是用于将 ZipServ 集成到**vLLM** [39]中的 Python 粘合代码。我们扩展了 vLLM 的模型加载器和线性执行模块，以支持 TCA-TBE 格式，并利用 PyBind11 调用自定义 CUDA 内核。

6 评估

我们从两个层面评估 ZipServ 的性能：融合 ZipGEMM 和独立解压内核（ZipServ-Decomp）的内核层，以及端到端推理框架层。所有实验均在两个平台上进行。❶ 一台消费级服务器，配备4× NVIDIA RTX4090 GPU（Ada Lovelace，24GB 显存，计算能力 8.9），并搭载 Intel Xeon Platinum 8352V CPU（144 核，512GB DDR4）。❷ 一个平台级数据中心，配备 4× NVIDIA L40S GPU（Ada Lovelace，48GB），并搭载 Intel Xeon Gold 6230R CPU（104 核，512GB DDR4）。我们还在最新的❸ RTX5090 GPU（Blackwell，32GB，计算能力 12.0）上评估 ZipGEMM，以展示其向前兼容性。所有代码均使用 GCC 11.3 和 NVCC 12.4 编译（RTX5090 专用 NVCC 12.8）。内核层评估中，我们先进行 100 次预热迭代，随后进行 1,000 次计时执行。端到端评估中，每种配置运行 10 次。

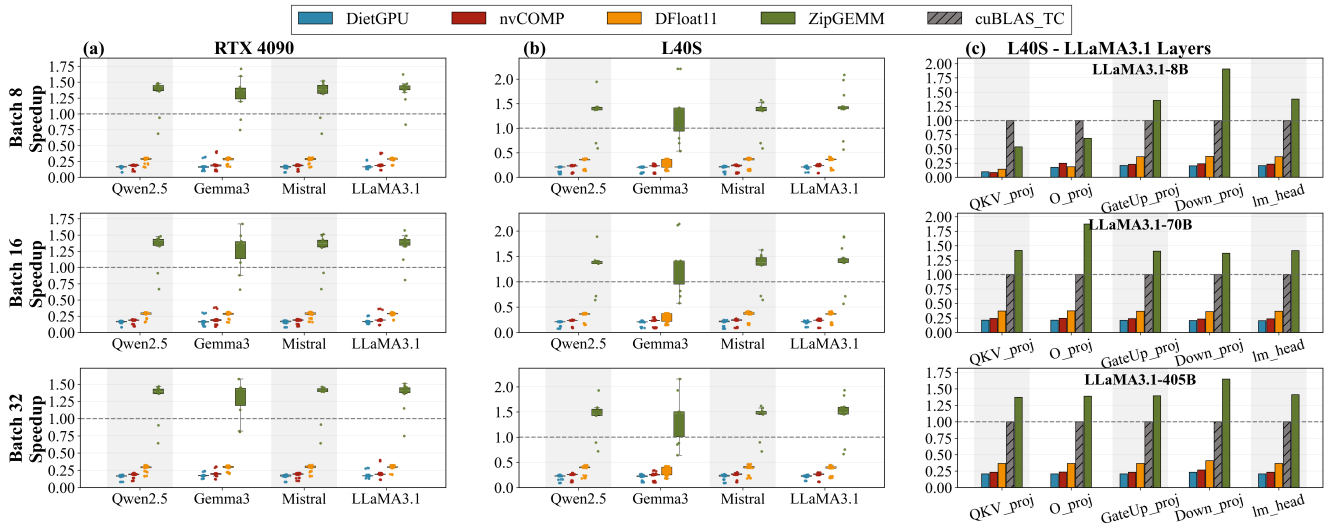

图 11。 NVIDIA RTX4090 和 L40S GPU 上的内核性能对比。

6.1 ZipGEMMKernel 性能

数据集。 我们使用当前先进大语言模型中的代表性线性层，对内核层性能进行基准测试。内核基准测试的输入形状直接从主流大语言模型系列的实际权重矩阵中提取，包括 LLaMA3。1 [17]（8B、70B 和 405B）、通义千问2。5 [69]（7B、14B、32B 和 72B）、Gemma3 [68]（12B 和 27B）以及 Mistral [2]（24B 和 123B），覆盖了广泛的模型规模和隐藏维度。

基线。 我们将 ZipGEMM 与四种具有代表性的基线进行比较：❶ cuBLAS_TC v12.4.5 [52], 英伟达官方的 BF16 张量核心通用矩阵乘法内核；❷ DietGPU [33], 一种流行的开源、GPU 原生 rANS 编解码器，用于浮点权重的无损解压缩；❸ nvCOMP (rANS) [53],英伟达基于非对称数制系统的通用解压缩库；以及❹ DFloat11 [85], 一种用于大语言模型推理的先进霍夫曼编码 GPU 解压缩框架。由于 nvCOMP 不支持原生 BF16，我们通过 rANS 将指数位压缩为比特流，并使用定制的高性能内核重建 BF16 值。对于 DFloat11，由于其压缩代码不可用，我们对完整 Transformer 块的解压缩延迟进行基准测试，并对其他矩阵形状的估算值进行线性缩放。

工作负载。 我们对 Transformer 模块中的所有线性层进行性能分析，包括合并的 QKV 投影（QKV_proj）、注意力输出投影（O_proj）、合并的 FFN 门控与上投影（GateUp_proj）和下投影（Down_proj），以及模型的语言模型头层。测试采用批量大小 8、16 和 32。

结果。我们首先评估融合式 *ZipGEMM* 内核的性能。图 11 展示了归一化相对于 cuBLAS_TC 的加速比，涵盖所有评估的模型和工作负载。ZipGEMM 在两个硬件平台上始终优于所有基线方法。在 RTX4090 上，ZipGEMM 相较于 cuBLAS_TC 的平均加速比为 1.31×，峰值加速比为 1.71×。在 L40S 上，优势更为显著，平均加速比为 1.36×，最高达到 2.21×。相比之下，其他解耦解压缩方法会引入大量开销，导致性能显著下降。具体而言，DietGPU、nvCOMP 和 DFloat11 在 RTX4090 和 L40S 上的平均加速比分别仅为 0.17 × /0.20×、 0.19 ×/0.23× 和 0.28 × /0.34× 。这表明，解耦解压缩过程产生的开销超过了基线通用矩阵乘法的计算时间。ZipGEMM 是唯一能够显著超越高效张量核心通用矩阵乘法的实现。这些结果凸显了 ZipGEMM 融合解压缩与计算方法的有效性；该方法能将存储节省高效转化为切实的执行加速。

我们进一步进行了逐层分析（图11(c)）。ZipGEMM 在 Transformer 块中大多数计算密集型层上表现出显著加速。例如，在 L40S 上的 LLaMA3.1 模型系列中，ZipGEMM 在 GateUp 1.39× 和 Down 1.64× proj_层上的平均加速比分别达到_。然而，在处理某些形状较小的层时，ZipGEMM 可能会出现性能下降；例如，在 L40S 上，其在 LLaMA3.1-8B 的 O_proj 层上的性能降至 0.79×。这主要是因为小型层需要进行细粒度参数调优（例如 split-K 配置和精确分块）才能充分利用硬件，这超出了本工作的范围。尽管如此，这类层仅占 Transformer 块内总浮点运算次数的一小部分。ZipGEMM 提供稳健的块级

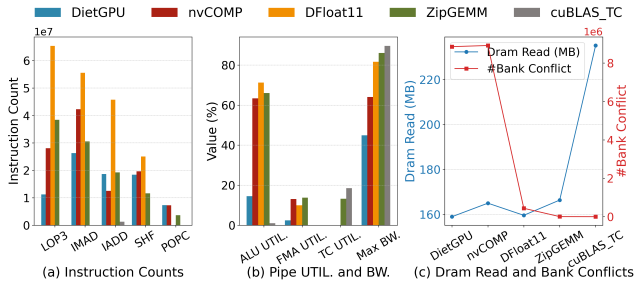

图 12. 微观级内核性能分析。

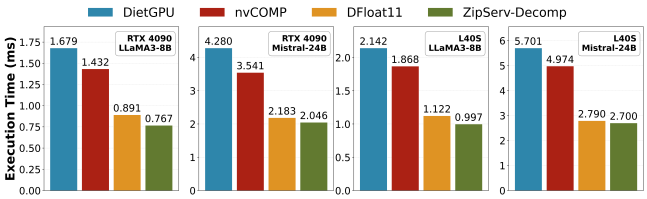

图 13. 独立解压内核比较。

在 L40S 上，LLaMA3.1-8B 的加速比为 1.35×，LLaMA3.1-405B 的加速比为 1.48×。

微观级分析。 我们使用 Nsight Compute (NCU) 在 RTX4090 上对 ZipGEMM 进行了性能分析，以确定其加速的来源（$M = 28672$，$K = 4096$ 和 $N = 32$）。如图 12所示，性能提升源于一种有意的架构权衡：通过引入可预测的 ALU 工作负载来即时解码，以换取内存流量的减少。图12(a) 对这一权衡进行了量化。大量的整数和逻辑指令（LOP3、IADD 和 POPC）反映了核心解码步骤的计算成本。这是 DRAM 读取量降低 29.3% 所付出的代价，直接验证了 TCA-TBE 格式的效率。至关重要的是，两级软件流水线通过将解码与计算和内存操作重叠，有效隐藏了解码延迟。因此，即使 ALU 利用率飙升至 66.0%，Tensor Core 利用率仍保持在 cuBLAS 基线的 71.6%，表现出色，证明计算吞吐量得以保留（图 12(b)）。这种高流水线效率得益于我们的数据布局。如图 12(c) 所示，与 DietGPU 等方法产生的数百万次共享内存组冲突相比，共享内存组冲突几乎完全消除（~4.7K）。这种无冲突访问是细粒度流水线的先决条件，可确保数据流畅传输并最大限度地提高 SIMT 吞吐量。

6.2 解压缩内核性能

为进一步剖析解压缩效率，我们对独立的ZipServ-解压缩内核进行了基准测试。图13 展示了 LLaMA3.1-8B 和 Mistral-24B 的完整 Transformer 块中所有权重的总解压缩时间。ZipServ- 解压缩的平均加速比分别达到 **2。14×**，**1。83×**，以及

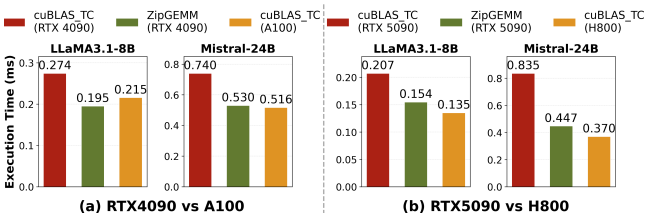

图 14. 跨代性能对比。

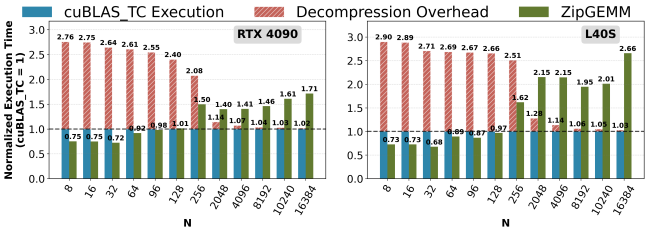

图 15. 不同 N 设置下 ZipServ 的性能。

1。10× 相较于 DietGPU、nvCOMP 和 DFloat11。尽管 TCA-TBE 格式在协同设计时旨在支持与矩阵乘法的融合执行，但事实证明，它在独立解压缩方面同样高效。这种效率源于其定长且与线程束对齐的设计，该设计消除了控制分歧，并支持线程束同步的逐线程解码。相比之下，尽管现有基线方案专门针对解压缩进行了优化，但它们通常依赖变长熵编码格式。这些格式会导致线程分歧、串行位解析和不规则内存访问，从而降低图形处理器效率。

6.3 不同图形处理器代际和层级

为确保前向兼容性，我们在最新的 NVIDIA RTX5090 上对 ZipGEMM 进行了基准测试，并使用 LLaMA3.1-8B 和 Mistral-24B 的 GateUp_投影层（批量大小为 32）与顶级数据中心 A100 和 H800 进行比较。首先，我们将 ZipGEMM 直接移植到基于 Blackwell 的 RTX5090 上，但不利用新特性（例如张量内存和异步 WMMA 执行 [32]）。如图 14所示，ZipGEMM 在 RTX5090 上显著快于 cuBLAS_TC：在 LLaMA3.1-8B 上提速 1.34× 倍，在 Mistral-24B 上提速 1.87× 倍，证实了该设计具备前向兼容性。ZipGEMM 也缩小了消费级与数据中心差距：在 RTX4090 上，使用 LLaMA3.1-8B 时，ZipGEMM 的性能优于 A100 上的标准 cuBLAS_TC（0.195 毫秒，而 0.215 毫秒，快 9.3%）；使用 Mistral-24B 时则仅慢 2.7%（0.530 毫秒，而 0.516 毫秒），实际上达到了同一性能级别。这一趋势在更新的硬件上更加明显。标准 RTX5090 的性能比 H800 低 53.3%（LLaMA3.1-8B）和 125.7%（Mistral-24B），而 ZipGEMM 将这一差距分别缩小至 14.1% 和 20.8%（图 14(b)），使消费级 GPU 接近数据中心级性能。

6.4 开销分析

我们从两个方面分析系统开销：运行时推理开销和离线准备成本。❶ 运行时开销。图 15 量化了 ZipServ 在不同 N 设置（$N = BS \times Seqlen$）下推理期间的开销。在解码阶段（较小的 N，通常为 1–128），融合 ZipGEMM 内核不会产生开销。相反，在这些受内存限制的场景中，它始终优于 cuBLAS_TC 基线，其即时解压完全隐藏在内核执行过程中。对于计算受限的预填充阶段（较大的 N，例如 8192），由于 ZipGEMM 即时解压的开销超过了减少内存访问所带来的收益，ZipServ会切换到解耦流水线。高效的解压缩内核会先展开压缩权重，随后再执行 cuBLAS_TC 通用矩阵乘法。这样只会产生有限的开销：在 ~4/16384 时，仅占通用矩阵乘法耗时的 %/2% N = 8192/16384。❷ 离线压缩成本。除了运行时性能，我们还评估了模型准备的一次性成本。在 16 核 Intel Xeon 8352V CPU 上压缩 LLaMA-3.1-8B 模型约需 2.5 分钟。由于这是部署前仅执行一次的离线操作，因此不会影响在线服务的关键路径；分摊到模型的整个生命周期后，这一成本可以忽略不计。

6.5 端到端推理性能

设置。 我们在一系列具有代表性的模型和硬件配置上评估 ZipServ 的端到端推理性能：在一块 RTX4090 GPU 上运行 LLaMA3.1-8B，在两块 L40S GPU 上运行 Mistral-24B，以及在四块采用张量并行的 L40S GPU 上运行 LLaMA3.1-70B。我们使用批量大小 8 和 32 进行基准测试，并设置 128、256、512、1024 和 2048 个词元的不同输出序列长度，以模拟不同的服务场景。我们将 ZipServ 与三个领先的基线系统进行比较： ❶ vLLM [39], 这一最先进的 LLM 推理和服务框架； ❷Transformers [75], 这一广泛采用的标准库；以及 ❸ DFloat11 [85], , 代表基于无损压缩的推理框架的最先进性能。我们测量两个关键指标：端到端请求延迟（生成完整输出序列所需的总时间）和吞吐量（每秒输出词元数）。如图 16所示，ZipServ 在所有测试配置下均持续展现出优异的性能。

结果在所有模型和批量大小上，平均而言，ZipServ 相较于 vLLM、Transformers 和 DFloat11，分别将**延迟**降低了 17.60%、60.79% 和 82.13%。在**吞吐量**方面，ZipServ 相较于 vLLM、Transformers 和 DFloat11，平均分别实现了 1.22× 、3.18× 和 8.52× 倍的加速。性能提升在长上下文生成任务中尤为显著，因为在解码阶段，融合 ZipGEMM 内核节省的内存带宽和计算效率开始发挥主导作用。例如，使用批量大小为 32 的配置生成 2048 个输出词元时

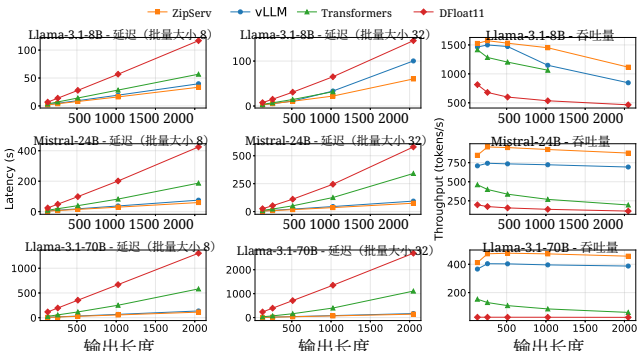

图 16. 端到端性能对比。

使用 LLaMA3.1-8B 时，ZipServ 的吞吐量达到 1105 令牌/秒，相较于 vLLM 实现了 1.66× 倍加速。我们还分析了推理期间的**内存消耗** 。对于 LLaMA3.1-8B、Mistral-24B 和 LLaMA3.1-70B，ZipServ 分别将权重占用空间从 14.96/43.92/131.56 GB 降至 10.83 (72.4%)/31.30 (71.3%)/93.52 (71.1%) GB。权重存储的减少还通过两种关键方式进一步提升了服务效率。首先，它使资源受限硬件能够部署更大的模型。其次，释放出的内存可分配给 KV 缓存，使 vLLM 的 PagedAttention 等内存管理器 [39]能够支持更大的批量大小和更长的上下文，从而将静态权重节省转化为动态吞吐量提升。

分解分析。 我们进一步分析了 LLaMA-3.1-8B 在 RTX4090 上的延迟和内存构成，以剖析性能提升的来源，详情见图 18。在基线 vLLM 系统中（序列长度为 1024），通用矩阵乘法操作占据了运行时间的主要部分，耗时 24.99 ms（占总延迟的 83.6%）。ZipServ 有效缓解了这一瓶颈：融合 ZipGEMM 内核结合残差稠密 GEMM，将线性层的总延迟降低至 14.76 ms，提升了 1。69× 由于注意力（3.02 ms）和其他开销（1.88 ms）保持不变，这些内核级性能提升直接带来了端到端加速。在内存方面，ZipServ 将静态权重从 14.96 GB 压缩至 11.18 GB。节省的 3.78 GB 空间由内存管理器自动重新分配，用于将 KV 缓存容量从 5.07 GB 扩大到 8.60 GB（增加了 1。70× ），从而实现端到端基准测试中观察到的更高吞吐量和更长上下文支持。

7 局限性与讨论

ZipServ 面向日益重要的部署场景而设计：在资源受限的消费级 GPU 和推理优化型 GPU 上，有限的内存带宽使无损压缩成为提升效率的有力手段。在这类平台上，ZipServ 始终能显著提升性能并节省内存。为了在带宽限制较宽松的条件下对性能进行压力测试，我们

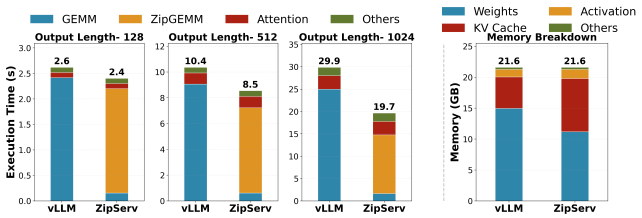

图 17. 端到端推理时间和内存消耗的细分情况。

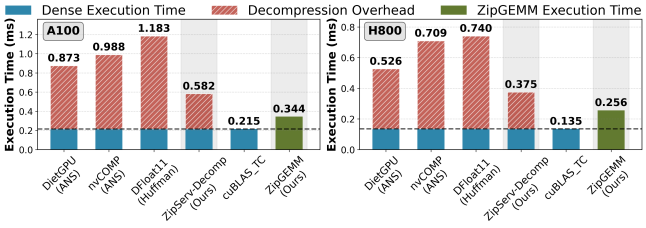

图 18. 面向训练的GPU上的性能。

还在面向训练的数据中心 GPU（A100、H800）上进行了基准测试。在这些 GPU 上，ZipGEMM 不一定总能达到高度优化的 cuBLAS 基线水平（图18）。这反映的是软硬件不匹配，而非算法限制：充足的高带宽内存（HBM2e/HBM3）缓解了 ZipServ 旨在解决的内存瓶颈，而较低的核心频率（例如，A100 为 1410 MHz，RTX4090 为 2520 MHz）则让密集的 ALU 工作负载更难在软件流水线中被隐藏。尽管如此，ZipServ仍为压缩推理提供了业界领先的支持。我们的独立解压缩内核性能最高可达当前最先进方法的 2 倍。64×此外，ZipGEMM 仍是最快的融合通用矩阵乘法内核。如 §6.3所示，ZipServ还能帮助消费级 GPU 缩小与顶尖数据中心加速器之间的大部分差距，为在易于获取的硬件上进行部署提供了极具吸引力的性价比优势。

虽然 ZipServ 的目标是实现位精确推理，但与有损技术进行比较也颇具参考价值。我们在 RTX4090 GPU 上，使用具有代表性的权重形状（28672×4096）和 32 的批量大小，对 ZipGEMM 与 Marlin W8A16 FP8 内核进行了基准测试。尽管 ZipGEMM 的延迟高于 Marlin-W8A16（0.194 毫秒对比 0.143 毫秒），但由此产生的 1.36× 差距与有效位宽之比（~11 位对比 FP8）非常接近。这表明，我们的设计降低了复杂无损解压缩的开销，并将其隐藏在内存访问延迟中。此外，ZipServ 与有损方法正交，可应用于量化权重之上以利用剩余冗余，将激进压缩与更高的性能相结合 [26]。

ZipServ 的扩展设想有三个关键方向。首先，TCA-TBE 格式可用于无损 KV 缓存压缩，以解决长上下文服务中的主要内存瓶颈 [45]。其次，尽管 ZipGEMM 目前针对 NVIDIA 架构进行了优化，但也可以适配其他矩阵加速器，包括 Intel AMX [37]和 AMD矩阵核心 [61]。核心设计与硬件无关，因此具备良好的可扩展性；解压缩所需的整数运算和人口计数指令在现代指令集中得到了广泛支持。最后，ZipServ 还适用于更广泛的系统级挑战，包括高效的模型检查点保存 [65, 71] 以及分布式训练中的通信压缩 [73, 84]。

8 相关工作

有损模型压缩。 有损方法主导着大语言模型加速，主要通过训练后量化（PTQ） [4, 12, 15, 19, 23, 43, 46, 78, 88] 和剪枝 [11, 14,22, 67, 80, 86], 并由高效内核支持 [21, 24, 55–57, 72, 77]。这些方法可能导致精度下降 [16, 41]。ZipServ 可实现位精确、无损且高效的推理。

无损模型压缩。 大量研究通过轻量级硬件方案进行内存压缩，以降低带宽需求或扩大容量 [7, 8, 20,38, 58, 59, 87], , 但这些技术并非针对模型压缩而设计。LMC [71] 和 ZipNN [29]等工作采用 Huffman [31] 压缩检查点，以便高效存储和分发，但无法带来运行时收益。包括 NeuZip 在内的近期系统 [28], DietGPU [33], nvCOMP [53],和 DFloat11 [85], 支持在运行时使用无损 GPU 编解码器来降低推理内存占用，但会产生显著开销（§3）。Huff-LLM [83] 效率更高，但面向类 FPGA 架构，无法推广至 GPU。Ecco [5]设计了专用的 Huffman 编解码硬件，但针对的是有损压缩。我们的 ZipServ将解压缩与通用矩阵乘法计算融合，使无损压缩成为实用的 GPU 推理加速方案。

算子融合。 算子融合通过组合算子来减少内存流量，例如 FlashAttention [9, 10, 62] 或图级框架 [48, 76, 79]。ZipServ 从中汲取经验，据我们所知，它首次将解压缩与通用矩阵乘法融合，从而避免了完整权重物化。

大语言模型推理的系统级优化。 现代大语言模型服务由复杂的推理引擎驱动 [1, 25, 27, 30, 39, 42, 64, 66, 82, 89, 90],, 这些引擎专注于高级调度策略和内存编排。ZipServ 与之正交且互为补充，可作为高性能后端集成其中。这样一来，这些引擎无需改变核心逻辑，即可同时受益于更低的内存占用和加速计算。

9 结论

我们推出了ZipServ，这是一种无损压缩框架，首次为大语言模型带来了显著的推理加速。ZipServ 通过协同设计硬件感知压缩格式 TCA-TBE 与融合解压缩-GEMM内核，突破了长期以来制约GPU上无损压缩方法的架构瓶颈。评估结果表明，与 cuBLAS 等经过高度优化的基准方案相比，ZipServ 实现了显著加速；尤其是在消费级硬件上，ZipServ 缩小了与昂贵数据中心GPU之间的性能差距，展现出令人信服的性价比优势。最终，ZipServ 将无损压缩重新定位为一种实用而强大的工具，使高性能、位精确的大语言模型推理成为可能，而不再只是节省存储空间的手段。

致谢

我们谨向匿名的 ASPLOS 评审以及指导委员会委员吴博致谢，感谢他们提供的宝贵意见与支持。本研究部分得到中国国家自然科学基金项目（项目编号：62272122）、广州市校企联合资助项目（项目编号：2024A03J0616）、广州市大数据智能重点实验室（2023 A03J0012）、香港协作研究基金项目（项目编号：C7004-22G 和 C6015-23G）、国家自然科学基金委员会／香港研究资助局联合科研计划（合同编号：CRS_HKUST601/24）以及中国国家自然科学基金项目（项目编号：62302126）的资助。Wei Wang 和 Xiaowen Chu 为通讯作者。

理论分析：大语言模型 BF16 权重的可压缩性

我们提出了理论基础，阐明了为什么大语言模型权重的指数分布高度偏斜，并呈现 top-K 连续性。

遵循近期研究 [13, 40, 63]，我们假设权重 $w \in \mathbb{R}^D$ 在单层中（为便于分析而向量化）服从零均值正态分布：

$$w \sim \mathcal{N}(0, \sigma^2 I)$$

一个非零的正态 BF16数 v 表示为 $v = (-1)^S \times 2^{E-127} \times (1.m_1 \ldots m_7)_2$，其中，$S$ 为符号位，E 为指数域的 8 位无符号整数值，而 $(1.m_1 \ldots m_7)_2$ 为隐含最高位 1 的 7 位尾数。BF16指数的偏置值为 127。

设 $x = E - 127$ 为实际的指数值。使用该特定指数 E 的任何数，其大小都在范围 $[2^x, 2^{x+1})$ 内。我们的分析聚焦于指数值的概率分布 x（或等价地，E），前提是权重 w 服从 $\mathrm{N}(0, \sigma^2)$。如果该分布 $P(X = x)$ 高度偏斜，就会产生冗余，这意味着某些指数值比其他指数值常见得多。

单个权重 w_i 落入对应于特定指数 x 的幅值范围的概率为：

$$P(X = x) = P(2^x \le |w_i| < 2^{x+1})$$

请注意，此计算为近似值。我们计算的是一个值落入指数理想幅值范围 $[2^x, 2^{x+1})$ 的概率，这通过忽略 7 位尾数造成的舍入效应，简化了 BF16 量化过程。不过，这为分析整体指数分布提供了可靠的近似。

给定 $w_i \sim \mathrm{N}(0, \sigma^2)$，其概率密度函数（PDF）为 $f(w_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-w_i^2/(2\sigma^2)}$。概率等于该 PDF 在正负区间上的积分：

$$P_\sigma(X = x) = 2 \times \int_{2^x}^{2^{x+1}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-t^2/(2\sigma^2)} dt$$

该积分可以用误差函数（erf）表示，其定义为 $\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-u^2} du$：

$$P_\sigma(X = x) = \mathrm{erf}\left(\frac{2^{x+1}}{\sigma\sqrt{2}}\right) - \mathrm{erf}\left(\frac{2^x}{\sigma\sqrt{2}}\right)$$

定理 A.1. 函数 $P(X = x) = erf\left(\frac{2^{x+1}}{\sigma\sqrt{2}}\right) - erf\left(\frac{2^x}{\sigma\sqrt{2}}\right)$ 在 $x \in \mathbb{Z}$ 上是单峰的。

证明。为证明其单峰性，我们考虑连续延拓 $f(x) = \mathrm{erf}\left(\frac{2^{x+1}}{\sigma\sqrt{2}}\right) - \mathrm{erf}\left(\frac{2^x}{\sigma\sqrt{2}}\right)$，其中 $x \in \mathbb{R}$。如果 $f(x)$ 是单峰的，那么离散函数 $P(X = x)$ 也将是单峰的；该函数是将 $f(x)$ 在整数点处取值后得到的。

令 $u = \frac{2^x}{\sigma\sqrt{2}}$，则 $f(x) = \mathrm{erf}(2u) - \mathrm{erf}(u)$。误差函数的导数为 $\frac{d}{dz}\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} e^{-z^2}$。因此，$f$ 关于 x 的导数为：

$$\frac{df}{dx} = \frac{2}{\sqrt{\pi}} u \ln 2 e^{-u^2} \left(2e^{-3u^2} - 1\right)$$

令 $h(u) = 2e^{-3u^2} - 1$。由于对于 $u > 0$（因为 $2^x > 0$），$\frac{2}{\sqrt{\pi}}$、u、$\ln 2$ 和 e^{-u^2} 均为正数，因此 $\frac{df}{dx}$ 的符号仅由 $h(u)$ 决定。

令 $h(u) = 0$，得到：

$$2e^{-3u^2} = 1 \implies e^{-3u^2} = \frac{1}{2} \implies -3u^2 = -\ln 2 \implies u^2 = \frac{\ln 2}{3}$$

因此，唯一的临界点为 $u_0 = \sqrt{\ln 2 \over 3}$。

对于 $u < u_0$，有 $3u^2 < \ln 2$，因此 $e^{-3u^2} > \frac{1}{2}$，这意味着 $h(u) > 0$ 且 $\frac{df}{dx} > 0$，所以 $f(x)$ 单调递增。

对于 $u > u_0$，我们有 $3u^2 > \ln 2$，因此 $e^{-3u^2} < \frac{1}{2}$，这意味着 $h(u) < 0$ 和 $\frac{df}{dx} < 0$，所以 $f(x)$ 是递减的。

因此， $f(x)$ 在 u_0处有唯一的最大值，这证明它是单峰的。由于 $P(X = x)$ 是 $f(x)$ 在整数值处的离散采样，因此 $P(X = x)$ 也是单峰的。

□

定理 A.2. 单峰分布中前 K 名的连续性。

证明。 反证法：假设概率最高的前 K 个值构成的集合X_K 不连续。那么，存在三个整数 $x_a < x_c < x_b$ 满足： x_a，$x_b \in \mathrm{X}_K$ 但$x_c \notin \mathrm{X}_K$。

根据单峰性质，概率函数 $P(x)$ 先递增后递减，因此对于介于 x_a和 x_b之间的任意 x_c， 我们有：

$$P(x_c) \geq \text{最小值}(P(x_a)\ P(x_b))。$$

由于 x_a和 x_b都在 X_K 中，因此它们属于概率最大的 K 个值。因此， $\min(P(x_a)\ P(x_b))$ 至少与第 K 大的概率一样大。因此， $P(x_c)$ 也必须至少与第 K 大的概率一样大，这意味着 x_c 应该在 X_K 中。

这与 $x_c \notin \mathrm{X}_K$ 的假设相矛盾。因此，前 K 名集合必须是连续的。□

参考文献

[1] Amey Agrawal、Nitin Kedia、Ashish Panwar、Jayashree Mohan、Nipun Kwatra、Bhargav S. Gulavani、Alexey Tumanov 和 Ramachandran Ramjee。2024年。利用 sarathi-serve 驯服大语言模型推理中的吞吐量与延迟权衡。载于 第*18*届 *USENIX* 操作系统设计与实现会议论文集 （美国加利福尼亚州圣克拉拉）（*OSDI'24*）。USENIX 协会，美国，论文编号 7，共 18 页。[2] Mistral AI。2023年。Mistral 7B。*arXiv* 预印本 *arXiv:2310.06825* （2023年）。[3] Zeyuan Allen-Zhu 和 Yuanzhi Li。2025年。《语言模型的物理学：第 3.3 部分，知识容量缩放定律》。载于 *ICLR*。OpenReview.net。[4] Saleh Ashkboos、Amirkeivan Mohtashami、Maximilian L Croci、Bo Li、Martin Jaggi、Dan Alistarh、Torsten Hoefler 和 James Hensman。2024年。Quarot：旋转大语言模型的无离群值 4 位推理。*arXiv* 预印本 *arXiv:2404.00456* （2024年）。[5] Feng Cheng、Cong Guo、Chiyue Wei、Junyao Zhang、Changchun Zhou、Edward Hanson、Jiaqi Zhang、Xiaoxiao Liu、Hai Li 和 Yiran Chen。2025年。Ecco：通过熵感知缓存压缩提升大语言模型的内存带宽和容量。载于 第*52*届计算机体系结构国际年会论文集。793–807。[6] Wei-Lin Chiang、Lianmin Zheng、Ying Sheng、Anastasios Nikolas An- gelopoulos、Tianle Li、Dacheng Li、Banghua Zhu、Hao Zhang、Michael I. Jordan、Joseph E. Gonzalez 和 Ion Stoica。2024年。Chatbot Arena：一个基于人类偏好评估大语言模型的开放平台。载于 *ICML*。OpenReview.net。[7] Esha Choukse、Mattan Erez 和 Alaa R Alameldeen。2018年。Compresso：实用的主存压缩。载于 *2018*年第*51*届 *IEEE/ACM* 微体系结构国际年会（*MICRO*）。IEEE，546–558。[8] Esha Choukse、Michael B Sullivan、Mike O'Connor、Mattan Erez、Jeff Pool、David Nellans 和 Stephen W Keckler。2020年。Buddy 压缩：为图形处理器上的深度学习和高性能计算工作负载扩展内存容量。载于 *2020*年 *ACM/IEEE* 第*47*届计算机体系结构国际年会（*ISCA*）。IEEE，926–939。[9] Tri Dao。2024年。FlashAttention-2：更快的注意力机制，具备更优的并行性和工作划分。载于国际学习表征会议（*ICLR*）。[10] Tri Dao、Daniel Y. Fu、Stefano Ermon、Atri Rudra 和 Christopher Ré。2022。FlashAttention：具备 I/O 感知能力的快速、节省内存的精确注意力计算。载于《神经信息处理系统进展》（NeurIPS）。[11] Rocktim Jyoti Das、Liqun Ma 和 Zhiqiang Shen。2023。超越规模：梯度如何影响大语言模型的剪枝决策。arXiv 预印本 arXiv:2311.04902（2023）。[12] Tim Dettmers、Mike Lewis、Younes Belkada 和 Luke Zettlemoyer。2022。Gpt3. int8 ()：面向大规模 Transformers 的 8 位矩阵乘法。《神经信息处理系统进展》35（2022），30318–30332。[13] Tim Dettmers、Artidoro Pagnoni、Ari Holtzman 和 Luke Zettlemoyer。2023。QLoRA：量化大语言模型的高效微调。载于 NeurIPS。[14] Peijie Dong、Lujun Li、Zhenheng Tang、Xiang Liu、Xinglin Pan、Qiang Wang 和 Xiaowen Chu。2024。Pruner-Zero：从零开始演化适用于大语言模型的符号化剪枝指标。载于第 41 届国际机器学习大会论文集。PMLR。https://arxiv.org/abs/2406.02924 [arXiv: 2406.02924].[15] Peijie Dong、Lujun Li、Yuedong Zhong、Dayou Du、Ruibo Fan、Yuhan Chen、Zhenheng Tang、Qiang Wang、Wei Xue、Yike Guo 等。2024。Stbllm：以结构化二进制大语言模型打破 1 位壁垒。arXiv 预印本 arXiv:2408.01803（2024）。[16] Peijie Dong、Zhenheng Tang、Xiang Liu、Lujun Li、Xiaowen Chu 和 Bo Li。2025。压缩大语言模型能真正行动吗？对大语言模型压缩中智能体能力的实证评估。arXiv 预印本 arXiv:2505.19433（2025）。[17] Abhimanyu Dubey、Abhinav Jauhri、Abhinav Pandey、Abhishek Kadian、Ahmad Al-Dahle、Aiesha Letman、Akhil Mathur、Alan Schelten、Amy Yang、Angela Fan 等。2024。Llama 3 模型家族。arXiv 预印本 arXiv:2407.21783（2024）。[18] Jarek Duda、Khalid Tahboub、Neeraj J Gadgil 和 Edward J Delp。2015。使用非对称数制系统准确替代霍夫曼编码。载于 2015 年图像编码研讨会（PCS）。IEEE，65–69。[19] Ali Edalati、Alireza Ghaffari、Mahsa Ghazvini Nejad、Lu Hou、Boxing Chen、Masoud Asgharian 和 Vahid Partovi Nia。2025。OAC：面向准确训练后量化的输出自适应校准。载于 AAAI。AAAI 出版社，16453–16461。[20] Magnus Ekman 和 Per Stenstro m。2005年。一种稳健的主存压缩方案。载于第 32 届国际计算机体系结构研讨会（ISCA'05）。IEEE，74–85。[21] Ruibo Fan、Xiangrui Yu、Peijie Dong、Zeyu Li、Gu Gong、Qiang Wang、Wei Wang 和 Xiaowen Chu。2025。SpInfer：利用底层稀疏性，在 GPU 上高效进行大语言模型推理。载于 EuroSys。ACM，243–260。[22] Elias Frantar 和 Dan Alistarh。2023。SparseGPT：可一次性准确剪枝大型语言模型。载于 ICML。[23] Elias Frantar、Saleh Ashkboos、Torsten Hoefler 和 Dan Alistarh。2022。Gptq：生成式预训练 Transformer 的准确训练后量化。arXiv 预印本 arXiv:2210.17323（2022）。[24] Elias Frantar、Roberto L. Castro、Jiale Chen、Torsten Hoefler 和 Dan Alistarh。2025。MARLIN：大语言模型上的混合精度自回归并行推理。载于 PPoPP。ACM，239–251。[25] Yao Fu、Leyang Xue、Yeqi Huang、Andrei-Octavian Brabete、Dmitrii Ustiugov、Yuvraj Patel 和 Luo Mai。2024。ServerlessLLM：面向大语言模型的低延迟无服务器推理。载于 OSDI。USENIX 协会，135–153。[26] Gerasi mos Gerogiannis、Stijn Eyerman、Evangelos Georganas、Wim Heirman 和 Josep Torrellas。2025。DECA：基于三维屋顶线模型的近核大语言模型解压缩加速器。载于第 58 届 IEEE/ACM 国际微体系结构研讨会论文集®。184–200。[27] Ruihao Gong、Shihao Bai、Siyu Wu、Yunqian Fan、Zaijun Wang、Xi-uhong Li、Hailong Yang 和 Xianglong Liu。2025。面向 SLA 保证下大语言模型服务的过去-未来调度器。载于第 30 届

ACM 编程语言与操作系统体系结构支持国际会议论文集，第 2 卷。798–813。[28] Yongchang Hao、Yanshuai Cao 和 Lili Mou。2024。NeuZip：通过神经网络动态压缩实现高效的内存使用、训练和推理。CoRR abs/2410.20650（2024）。[29] Moshik Hershcovitch、Andrew Wood、Leshem Choshen、Guy Girmon-sky、Roy Leibovitz、Ilias Ennmouri、Michal Malka、Peter Chin、Swami-nathan Sundararaman 和 Danny Harnik。2024。ZipNN：人工智能模型无损压缩。CoRR abs/2411.05239（2024）。[30] Connor Holmes、Masahiro Tanaka、Michael Wyatt、Ammar Ahmad Awan、Jeff Rasley、Samyam Rajbhandari、Reza Yazdani Aminabadi、Heyang Qin、Arash Bakhtiari、Lev Kurilenko 和 Yuxiong He。2024。DeepSpeed-FastGen：通过 MII 和 DeepSpeed-Inference 实现大语言模型的高吞吐量文本生成。arXiv:2401.08671 [cs.PF] https://arxiv. org/abs/ 2401 08671 。[31] David A Huffman。2007。一种构造最小冗余码的方法。《IRE 论文集》40，9（2007），1098–1101。[32] Aaron Jarmusch、Nathan Graddon 和 Sunita Chandrasekaran。2025。通过微基准测试剖析 NVIDIA Blackwell架构。arXiv 预印本 arXiv:2507.10789（2025）。[33] Jeff Johnson。2024。DIET-GPU：在 GPU 上高效进行模型推理。https://github.com/facebookresearch/dietgpu。[34] Norm Jouppi、George Kurian、Sheng Li、Peter Ma、Rahul Nagarajan、Lifeng Nai、Nishant Patil、Suvinay Subramanian、Andy Swing、Brian Towles 等。2023。Tpu v4：一台用于机器学习、支持嵌入且可通过光学方式重新配置的超级计算机。载于第 50 届年度国际计算机体系结构研讨会论文集。1–14。[35] Dhiraj Kalamkar、Dheevatsa Mudigere、Naveen Mellempudi、Dipankar Das、Kunal Banerjee、Sasikanth Avancha、Dharma Teja Vooturi、Nataraj Jammalamadaka、Jianyu Huang、Hector Yuen 等。2019。BFloat16 在深度学习训练中的应用研究。arXiv 预印本 arXiv:1905.12322（2019）。[36] Jared Kaplan、Sam McCandlish、Tom Henighan、Tom B. Brown、Benjamin Chess、Rewon Child、Scott Gray、Alec Radford、Jeffrey Wu 和 Dario Amodei。2020。神经语言模型的缩放定律。CoRR abs/2001.08361（2020）。[37] Hyungyo Kim、Gaohan Ye、Nachuan Wang、Amir Yazdanbakhsh 和 Nam Sung Kim。2024。利用英特尔高级矩阵扩展（AMX）进行大语言模型推理。《IEEE 计算机体系结构快报》23，1（2024），117–120。[38] Jungrae Kim、Michael Sullivan、Esha Choukse 和 Mattan Erez。2016。位平面压缩：转换数据以改善众核架构中的压缩效果。《ACM SIGARCH 计算机体系结构新闻》44，3（2016），329–340。[39] Woosuk Kwon、Zhuohan Li、Siyuan Zhuang、Ying Sheng、Lianmin Zheng、Cody Hao Yu、Joseph Gonzalez、Hao Zhang 和 Ion Stoica。2023。通过 PagedAttention 为大语言模型服务实现高效内存管理。载于 SOSP。ACM，611–626。[40] Hoil Lee、Fadhel Ayed、Paul Jung、Juho Lee、Hongseok Yang 和 François Caron。2023。具有依赖权重的深度神经网络：高斯过程混合极限、重尾、稀疏性与可压缩性。《机器学习研究期刊》24（2023），289:1–289:78。[41] Zhen Li、Yupeng Su、Runming Yang、Zhongwei Xie、Ngai Wong 和 Hongxia Yang。2025。量化与推理相遇：探索大语言模型低比特量化对数学推理的退化影响。CoRR abs/2501.03035（2025）。[42] Zhuohan Li、Lianmin Zheng、Yinmin Zhong、Vincent Liu、Ying Sheng、Xin Jin、Yanping Huang、Zhifeng Chen、Hao Zhang、Joseph E. Gonzalez 和 Ion Stoica。2023。AlpaServe：通过模型并行和统计复用实现深度学习服务。载于 OSDI。USENIX 协会，663–679。[43] Ji Lin、Jiaming Tang、Haotian Tang、Shang Yang、Wei-Ming Chen、Wei-Chen Wang、Guangxuan Xiao、Xingyu Dang、Chuang Gan 和 Song Han. 2024. AWQ：面向设备端大语言模型压缩与加速的激活感知权重量化。机器学习与系统会议论文集 6 (2024), 87–100.[44] Ruikang Liu, Yuxuan Sun, Manyi Zhang, Haoli Bai, Xianzhi Yu, Tiezheng Yu, Chun Yuan, and Lu Hou. 2025. 量化会损害推理能力吗？关于量化推理模型的实证研究。CoRR abs/2504.04823 (2025).[45] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Anantha-narayanan, et al. 2024. Cachegen：面向大语言模型快速服务的 KV 缓存压缩与流式传输。载于 ACM SIGCOMM 2024 会议论文集。38–56.[46] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. 2024. SpinQuant——采用学习得到的旋转进行大语言模型量化。arXiv 预印本 arXiv:2405.16406 (2024).[47] Weile Luo, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. 2024. 英伟达 Hopper GPU 架构的基准测试与剖析。载于 2024 年 IEEE 国际并行与分布式处理研讨会 (IPDPS)。IEEE, 656–667.[48] Lingxiao Ma, Zhiqiang Xie, Zhi Yang, Jilong Xue, Youshan Miao, Wei Cui, Wenxiang Hu, Fan Yang, Lintao Zhang, and Lidong Zhou. 2020. Rammer：通过 rTasks 实现全面的深度学习编译器优化。载于第14届 USENIX 操作系统设计与实现研讨会 (OSDI 20)。USENIX 协会，881–897。https: //www.usenix.org/conference/osdi20/presentation/ma[49] Anmol Mekala, Anirudh Atmakuru, Yixiao Song, Marzena Karpinska, and Mohit Iyyer. 2025. 量化是否会影响模型在长上下文任务中的性能？arXiv 预印本 arXiv:2505.20276 (2025). [50] 英伟达。2020。NVIDIA Ampere GA102 GPU 架构白皮书。https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf。[51] 英伟达。2023。NVIDIA Ada GPU 架构白皮书。https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf。[52] 英伟达。2024。cuBLAS 文档。https://docs.nvidia.com/cuda/cublas/index.html.[53] 英伟达。2025。nvcomp：nvCOMP 文档与示例代码仓库。https://github.com/NVIDIA/nvcomp。访问日期：2025-08-18。[54] OpenAI。2023。GPT-4 技术报告。arXiv:2303.08774 [cs.CL][55] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. LUT-GEMM：基于查找表、面向大规模生成式语言模型高效推理的量化矩阵乘法。载于 ICLR。OpenReview.net.[56] Gunho Park, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2022. nuQmm：面向大规模生成式语言模型高效推理的量化矩阵乘法。CoRR abs/2206.09557 (2022)。[57] Tommaso Pegolotti, Elias Frantar, Dan Alistarh, and Markus Püschel. 2023. QIGen：为大语言模型量化推理生成高效内核。CoRR abs/2307.03738 (2023).[58] Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2013. 线性压缩页：一种低复杂度、低延迟的主存压缩框架。载于第46届 IEEE/ACM 国际微体系结构研讨会论文集。172–184.[59] Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Phillip B Gibbons, Michael A Kozuch, and Todd C Mowry. 2012. 基值-增量-立即数压缩：面向片上缓存的实用数据压缩。载于第21届并行架构与编译技术国际会议论文集。377–388。[60] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer：语言模型能够自学

如何使用工具。载于 NeurIPS。[61] Gabin Schieffer, Daniel Araújo De Medeiros, Jennifer Faj, Aniruddha Marathe, and Ivy Peng. 2024. AMD矩阵核心的兴起：性能、能效与可编程性。载于 2024 IEEE 国际系统与软件性能分析研讨会 (ISPASS)。IEEE, 132–143.[62] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ra-mani, and Tri Dao. 2024. FlashAttention-3：兼具速度与精度的注意力计算，支持异步性与低精度。载于 NeurIPS。[63] Chongjie Si, Jingjing Jiang, and Wei Shen. 2025. 揭示大型基础模型中权重的奥秘：高斯分布永不消退。CoRR abs/2501.10661 (2025).[64] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer：利用消费级 GPU 实现大语言模型快速服务。载于 ACM SIGOPS 第 30 届操作系统原理研讨会论文集（美国得克萨斯州奥斯汀）（SOSP '24）。计算机协会，纽约，美国，

590 606 10 1145 3694715 3695964 – 。 2025 [65] doi: 。/ 。Foteini Strati, Michal Friedman, and Ana Klimovic. 。PCcheck：面向机器学习的持久化并发检查点。载于第 30 届 ACM 编程语言与操作系统架构支持国际会议论文集，第 1 卷。811–827.[66] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix：大语言模型服务的动态调度。载于 OSDI。USENIX 协会，173–191。[67] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. 一种简单有效的大语言模型剪枝方法。载于 ICLR。[68] Gemma 团队。2025。Gemma 3 技术报告。arXiv 预印本 arXiv:2503.19786 (2025)。[69] Qwen 团队。2024。Qwen2.5 技术报告。arXiv 预印本 arXiv:2412.15115 (2024)。[70] Qwen 团队。2025。Qwen3 技术报告。arXiv 预印本 arXiv:2505.09388 (2025).[71] Daniel Waddington and Cornel Constantinescu. 2025. 大语言模型张量增量快照的无损压缩。arXiv 预印本 arXiv:2505.09810 (2025).[72] Lei Wang, Lingxiao Ma, Shijie Cao, Quanlu Zhang, Jilong Xue, Yining Shi, Ningxin Zheng, Zi ming Miao, Fan Yang, Ting Cao, et al. 2024. Ladder：通过硬件感知张量变换实现高效低精度深度学习计算。载于第18届 USENIX 操作系统设计与实现研讨会 (OSDI 24)。307–323。[73] Zhuang Wang, Zhaozhuo Xu, Jingyi Xi, Yuke Wang, Anshumali Shri-vastava, and TS Eugene Ng. 2025. {ZEN}：通过稀疏性驱动的数据同步赋能分布式训练。载于第19届 USENIX 操作系统设计与实现研讨会 (OSDI 25)。537–556。[74] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. 思维链提示促使大语言模型进行推理。载于 NeurIPS。[75]Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. 2020. Transformers：自然语言处理领域的先进技术。载于 2020 年自然语言处理实证方法会议：系统演示论文集。38–45。[76] Mengdi Wu, Xinhao Cheng, Shengyu Liu, Chunan Shi, Jianan Ji, Kit Ao, Praveen Velliengiri, Xupeng Miao, Oded Padon, and Zhihao Jia. 2025. Mirage：面向张量程序的多层级超优化器。载于第19届 USENIX 操作系统设计与实现研讨会 (OSDI 25)。USENIX 协会。h
tt
ps://www.usenix.org/conference/osdi25/presentation/wu-mengdi [77] Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. 2023. Flash-LLM：实现经济高效且高性能的大规模生成

非结构化稀疏性模型推理。《VLDB 汇刊》17卷第2期（2023年10月），211–224。doi:10.14778/3626292.3626303[78] Guangxuan Xiao、Ji Lin、Mickael Seznec、Hao Wu、Julien Demouth 和 Song Han。2023年。Smoothquant：大语言模型训练后量化的准确高效方法。载于《国际机器学习大会》。PMLR，38087–38099。[79] Jiarong Xing、Leyuan Wang、Shang Zhang、Jack Chen、Ang Chen 和 Yibo Zhu。2022年。Bolt：弥合自动调优器与硬件原生性能之间的差距。机器学习与系统会议论文集4（2022年），204–216。[80] Peng Xu、Wenqi Shao、Mengzhao Chen、Shitao Tang、Kaipeng Zhang、Peng Gao、Fengwei An、Yu Qiao 和 Ping Luo。2024年。BESA：通过分块参数高效稀疏分配剪枝大语言模型。载于 ICLR。[81] Tian Ye、Zicheng Xu、Yuanzhi Li 和 Zeyuan Allen-Zhu。2025年。《语言模型的物理学：第2.2部分，如何从小学数学题的错误中学习》。载于 ICLR。OpenReview.net。[82] Gyeong-In Yu、Joo Seong Jeong、Geon-Woo Kim、Soojeong Kim 和 Byung-Gon Chun。2022年。Orca：面向基于 Transformer 的生成模型的分布式服务系统。载于第16届 USENIX 操作系统设计与实现研讨会（OSDI 22）。USENIX 协会，卡尔斯巴德，加利福尼亚州，521–538。
https://www.usenix.org/conference/ osdi22/presentation/yu[83] Patrick Yubeaton、Tareq Mahmoud、Shehab Naga、Pooria Taheri、Tianhua Xia、Arun George、Yasmein Khalil、Sai Qian Zhang、Siddharth Joshi、Chinmay Hegde 和 Siddharth Garg。2025年。Huff-LLM：面向高效大语言模型推理的端到端无损压缩。arXiv:2502.00922 [cs.LG] https://arxiv.org/abs/2502.00922[84] Lin Zhang、Lo ngteng Zhang、Shaohuai Shi、Xiaowen Chu 和 Bo Li。2023年。分布式深度学习中梯度压缩的评估与优化。载于2023年 IEEE 第43届国际分布式计算系统会议（ICDCS）。IEEE，361–371。[85] Tianyi Zhang、Yang Sui、Shaochen Zhong、Vipin Chaudhary、Xia Hu 和 Anshumali Shrivastava。2025年。70% 的体积，100% 的准确率：借助动态长度浮点数实现面向高效图形处理器推理的无损大语言模型压缩。arXiv 预印本 arXiv:2504.11651（2025年）。[86] Yingtao Zhang、Haoli Bai、Haokun Lin、Jialin Zhao、Lu Hou 和 Carlo Vittorio Cannistraci。2024年。即插即用：一种高效的大语言模型训练后剪枝方法。载于第十二届国际学习表征会议。[87] Jishen Zhao、Sheng Li、Jichuan Chang、John L Byrne、Laura L Ramirez、Kevin Lim、Yuan Xie 和 Paolo Faraboschi。2015年。Buri：借助基于硬件的内存扩展，拓展大内存计算规模。ACM 架构与代码优化汇刊（TACO）12卷第3期（2015年），1–24。[88] Yilong Zhao、Chien-Yu Lin、Kan Zhu、Zihao Ye、Lequn Chen、Size Zheng、Luis Ceze、Arvind Krishnamurthy、Tianqi Chen 和 Baris Kasikci。2024年。Atom：面向高效、准确大语言模型服务的低比特量化。机器学习与系统会议论文集6（2024年），196–209。[89] Lianmin Zheng、Liangsheng Yin、Zhiqiang Xie、Chuyue Sun、Jeff Huang、Cody Hao Yu、Shiyi Cao、Christos Kozyrakis、Ion Stoica、Joseph E. Gonzalez、Clark Barrett 和 Ying Sheng。2025年。SGLang：结构化语言模型程序的高效执行。载于神经信息处理系统大会第38届国际会议论文集（加拿大不列颠哥伦比亚省温哥华）（NIPS '24）。Curran Associates Inc.，美国纽约州雷德胡克，文章编号2000，共27页。[90] Yinmin Zhong、Shengyu Liu、Junda Chen、Jianbo Hu、Yibo Zhu、Xuanzhe Liu、Xin Jin 和 Hao Zhang。2024年。DistServe：将预填充与解码分离，实现面向有效吞吐量优化的大语言模型服务。载于第18届 USENIX 操作系统设计与实现研讨会（OSDI 24）。USENIX 协会，圣克拉拉，加利福尼亚州，193–210。
htt
ps://www.usenix.org/conference/osdi24/presentation/zhong-yinmin