arXiv:2611.33080v1 [cs.LG] 24 Nov 2026

移位量化：面向原生整数计算的紧凑表示

Yufeng Huang
yufeng.huang@columbia.edu

摘要

分组量化用于大语言模型量化，可在低位宽下提升准确率，但对权重进行分组会使标准整数矩阵乘加无法覆盖完整点积，从而影响高效的 GPU 推理。我们提出了一种新的量化方法——移位量化（SQ），通过追加并移位共享位来重构原生位宽整数操作数。为进一步提升效率，我们开发了 SQ 的一种特殊变体——移位整数（SHI），以使用 INT4 内核。我们表明，与标准分组量化相比，SQ 和 SHI 格式在质量和推理速度方面表现更佳：质量通过投影误差和 KL 散度衡量，推理速度则通过解码和预填充过程中的表现衡量。对于 2 位量化等低精度场景，SQ2 和 SHI2 在 Q2_K 检查点上实现了 llama.cpp 的 4,096 个令牌预填充吞吐量的 2.71–3.77× ，并将 WikiText-2 KL 散度降低 6.6–7.8%。对于 4 位量化等中等精度场景，SQ4 在所有测试模型和提示长度下，实现了 NInfer 分组整数基线预填充吞吐量的 1.43–2.47× ，同时 KL 散度比 NInfer 基线低 24–40%。对于 8 位量化等高精度场景，SQ8 的批量为一的预填充吞吐量比 vLLM 的在线 FP8 高 27.4%，KL 散度低 72.7%。

结果一览

移位格式	基线	预填充吞吐量	相对 KL 改善
2 位 SQ2 和 SHI2	llama.cpp Q2_K	2。71–3.77×	7–9%
4 位 SQ4	NInfer 分组整数	1.43–2.47×	31–65%
8 位 SQ8	vLLM 在线 FP8	高出 27.4%	266%

相对 KL 改善为 $100(\mathrm{KL}_{\mathrm{baseline}}/\mathrm{KL}_{\mathrm{Shifter}} - 1)\%$。

1 引言

随着大语言模型不断发展，前沿模型的参数量如今已超过一万亿 [5,17, 19, 27]。本地推理通常使用较小的模型，但消费级和工作站 GPU 仍面临严格的内存预算。模型权重与注意力键值（KV）缓存共用这部分内存；而缓存会随着上下文长度和活跃请求数量的增加而增长 [12, 16]。

对权重和键值缓存进行量化，有助于在本地硬件上运行更大的模型并支持更长的上下文 [6,16]。然而，在低位宽下，量化必须在大权重所需的取值范围与小权重所需的分辨率之间取得平衡。分组缩放通过为数值组分配缩放因子来应对这一权衡

。这种方法用于 GPTQ 和 AWQ 等学术方法、llama.cpp 的 Qx_0、Qx_K 和 IQx 系列 [9], ，也用于硬件支持的格式，例如 NVIDIA Blackwell 上的 NVFP4 和 MXFP4，以及 AMD CDNA 上的 MXFP4 4 [2, 23–25]。

分组量化也会影响推理速度。当点积跨越具有不同缩放因子的组时，内核必须分别保留每个组的贡献，以便应用相应的缩放因子 [15, 34]。这会阻止传统整数矩阵乘累加（MMA）指令在整个归约维度上进行累加。内核可以在每个组内使用 MMA，但较小的组限制了其对硬件的利用效率 [15, 34]。近期的 GPU 提供了针对有限格式和组大小的块缩放指令 [2, 23]。如何高效支持其他分组表示仍然是一项挑战。

为了解决这一限制，我们提出移位量化（SQ），通过支持在整个归约维度上执行 MMA，提高低比特本地推理的效率。在乘法之前，我们利用廉价的位移操作，将以低精度存储在 GPU寄存器中的权重展开。我们将这些操作应用于四个或八个权重组成的组（G4 或 G8），其规模小于传统量化中常用的 G16、G32 或 G64 组。较小的组能够以更细的粒度适应权重变化，从而减少量化误差。SQ 保留了小组的精细适应能力，同时允许整数矩阵乘加覆盖整个归约维度。

在 SQ 的基础上，我们开发了移位整数（SHI），以利用更快的 INT4 内核。SHI 将主乘积保留为 INT4，并在需要时通过较短的乘积应用分组校正。得益于这种特殊格式，分组校正只增加一小部分W4A4开销。

- **移位量化（SQ）**。 SQ 为每组存储低比特权重编码及共享信息。它通过附加共享位、应用组特定移位并截断至目标位宽，来重构整数操作数。重构后的操作数可在整个归约维度上执行整数矩阵乘加，之后再应用通道缩放。

- **移位整数（SHI）**。 SHI 基于 SQ 构建，使用 INT4 进行主矩阵乘积计算。它通过对组和进行额外的整数乘积计算来表示组级权重校正。这些乘积比主乘积短，也可以计入激活校正。

- **实现与评估**。 我们在消费级GPU上评估 SQ 和 SHI 的本地推理性能，并在投影、全模型和服务层面测量准确率与速度。我们的对比涵盖 llama.cpp 量化格式 [9], NVFP4，以及由社区开发的 NInfer 服务引擎 [20]。

第 3 节定义了表示和执行配置。第 4 节单独考察一个 MLP 投影，以展示不同执行路径对延迟和误差的影响。第 5 节将完整检查点与 BF16 参考结果进行比较，并测量预填充性能；第 6 节则比较四比特服务系统在预填充和推测解码方面的表现。

2 背景与相关工作

分组缩放为每个组设定适合其数值范围的缩放因子。这一理念早于神经网络，曾用于数字滤波器的块浮点运算 [26]。VS-Quant 研究了适用于神经网络推理的更细粒度缩放 [4]。较小的组能更准确地表示局部范围，但也需要更多缩放元数据，并使内核更频繁地应用缩放因子。

若干低比特方法采用分组或块级缩放。GPTQ 和 AWQ 支持分组整数权重量化 [8, 14]; ZeroQuant 将细粒度量化与优化推理内核相结合 [33]。QLoRA 的 NF4 将块级缩放因子与非均匀元素值结合起来 [7]„ 而 llama.cpp 则为低比特推理提供 Q 和 IQ 格式 [9]。这些格式将存储空间分配给权重编码、缩放因子及其他元数据。我们在投影基准测试中报告每权重打包比特数，以及准确率和速度。

组缩放因子也会影响 GPU 计算点积的方式。对于权重编码 q_k、组缩放因子s_g和激活值 ξ_k，输出为 $\sum_g s_g \sum_{k\in g} q_k \xi_k$。当不同组的缩放因子不同时，必须先对各个部分和进行缩放，再将结果合并。Atom 使用整数矩阵乘加计算分组乘积，并在内核中应用其缩放因子 [34]。QServe 指出，权重和部分和反量化是低比特 GPU 推理中显著的开销来源 [15]。SQ 则将组信息纳入重构整数操作数。SHI 将分组校正表示为基于组和的额外整数乘积。

块缩放浮点格式将局部缩放因子与硬件支持的算术运算相结合 [28]。MXFP4 和 MXFP8 对 32 个元素共用一个指数缩放因子 [25, 29]。NVFP4 为每 16 个四位元素使用一个 FP8 缩放因子，并使用一个全局缩放因子 [24]。最新的 GPU 指令支持特定的块缩放格式和组大小 [2, 23]。由于 SHI4 和 NVFP4 的权重存储占用相近，我们比较它们的准确率和速度。

异常值会增加低比特量化的难度。LLM.int8() 会将异常特征隔离出来，以便进行高精度计算 [6]。SmoothQuant 会在激活通道和权重通道之间重新分配量化难度 [32]。QuaRot 使用旋转将异常值分散到各个通道 [3]。我们使用通道缩放因子和带符号哈达玛旋转来调整输入和权重。随后，SQ 和 SHI 决定变换后的权重如何存储，以及如何用于整数计算。

3 量化格式与执行

3.1 线性投影与量化基

考虑一个线性 投影，其中激活值按行排列 $X \in \mathbb{R}^{M\times K}$，权重为 $W \in \mathbb{R}^{N\times K}$:

$$Y = XW^{\top}. \tag{1}$$

这里， M 表示输入词元数量， K 表示归约维度， N 表示输出宽度。我们对线性投影进行量化，同时保留外围的注意力机制、循环状态和非线性计算。不同的投影可以采用不同的表示；因此，分配给 MLP 组的位宽并不代表整个模型的平均精度。

我们使用正交变换 U 和对角通道缩放因子 C_i 与 C_o 对每个投影进行条件化。拟合整数重构 P，使得

$$WU \simeq C_o P C_i, \qquad Y \simeq (XUC_i)P^\top C_o. \tag{2}$$

第一个关系定义了拟合权重基，第二个关系由 $UU^\top = I$ 得出。输入通道缩放因子作用于收缩运算之前；输出通道缩放因子作用于其之后。独立拟合的投影即使共享同一输入，也可能具有不同的 C_i，因此融合执行会保留各自独立的缩放向量。

我们的分块变换是一种带符号归一化哈达玛旋转。对于包含 $B = 512$ 个输入通道的分块，记作

$$U_B = D_B H_B, \qquad H_B H_B^\top = I, \tag{3}$$

其中，D_B 包含符号，而 H_B 是按自然二进制顺序排列的归一化西尔维斯特哈达玛矩阵。完整变换 U 为分块对角矩阵。对于按行组织的输入，依次应用符号、哈达玛蝶形变换和输入通道缩放；逆变换则反转变换顺序。显式存储符号可确保该旋转在不同后端间可复现。我们在线性边界处应用该旋转，不将其移过非线性运算。旋转是一种成熟的条件化技术 [3]；它对量化基进行条件化，且独立于 SQ 和 SHI 编码构造。

在整个方法中，我们区分存储编码、重构操作数和缩放权重。三位编码可扩展为 INT8 操作数，同时保持紧凑的存储形式，但保留扩展后的数据组会增加驻留内存。因此，内存核算会分别统计编码位、通道参数、变换数据、预处理权重平面和临时工作区。

3.2 SQ整数重构

SQ 通过三种位运算将紧凑编码扩展为 INT4 或 INT8 操作数：追加共享位、应用分组移位，以及截断至目标位宽。这些整数运算在乘法之前融入组级信息，取代浮点分组重构。完整流程为

$$\begin{aligned} z &= (q \ll m) + a_g \\ v &= (z \ll \ell) \gg r_g \\ P &= \mathrm{clip}_t(v \gg \delta) \end{aligned}$$

(1) 追加共享位，(2) 应用分组移位，(3) 截断至目标位宽。

此处，q 是有符号基础编码，a_g 和 r_g 是共享组字段，t 是原生操作数位宽。符号 $\ll$ 和 $\gg$ 分别表示整数左移和算术右移，有符号舍入定义如下。固定对齐 ℓ 和最终缩减 δ 决定保留的位位置；它们不会增加学习得到的组字段。追加会扩展编码，移位提供局部二次幂自适应，而截断则生成目标位宽的操作数。因此，局部自适应被纳入整数预处理过程，使通道缩放和输出转换位于整数缩减之外。

我们将一种 SQ 格式命名为 SQx-GyAmSn：x 是每个基础编码的位宽 q，y 是每组的权重数，m 是共享追加字段的位宽a_g，n 是组移位选择器的位宽r_g。例如，SQ2-G4A1S1 使用两位编码，每组包含四个权重，并为每组配备一个追加位和一个一位移位选择器。下面介绍这三种操作。

3.2.1 追加共享位

设 $q_{j,\ k}$ 是输出通道 j 和输入通道 k对应的有符号 x 位编码。将每个权重行划分为若干组，每组包含 y 个连续的输入通道，其中 $g(k) = \lfloor k/y \rfloor$。每组存储一个 m 位追加字段$a_{j,\ g}$。扩展整数为

$$z_{j,k} = 2^m q_{j,k} + a_{j,g(k)}, \quad \begin{cases} -2^{x-1} \le q_{j,k} < 2^{x-1}, \\ 0 \le a_{j,g} < 2^m. \end{cases} \tag{4}$$

左移会腾出 m 个低位位置，用于填入共享字段。这样无需为每个元素存储所有追加位，即可扩展每个编码。例如，带有四个追加位的 INT4 基值覆盖 INT8 范围，而在 $y = 8$ 个权重之间共享这些位，每个权重仅需增加 $4/8 = 0.5$ 位存储。移位字段和通道参数分别占用存储空间。

3.2.2 应用分组移位

共享选择器$r_{j,\ g}$ 可将扩展后的编码定位到操作数中。在整数舍入之前，右移 r 位会将数值缩放 2^{-r}：右移一位、两位和三位分别对应因子 $1/2$、$1/4$ 和 $1/8$。因此，分组移位通过整数指令而非浮点乘法实现局部缩放适配。

3.2.3 截断至硬件宽度

最后一步是将定位后的值缩减为有符号的 t 位操作数。定义

$$R_d(v) = \left\lfloor v/2^d \right\rfloor, \qquad d \ge 0, \tag{5}$$

如算术右 移位，包括对负输入的舍入。将这三个步骤组合起来可得 s

$$P_{j,k} = \mathrm{clip}_t\Big(R_\delta\Big(R_{r_{j,g(k)}}(2^\ell z_{j,k})\Big)\Big), \tag{6}$$

其中 $\ell \ge 0$ 指定对齐方式，$\delta \ge 0$ 指定最终缩减，而 clip_t 会在需要时应用饱和。这些参数和饱和规则都是表示的一部分。当所有缩减后的值都在可表示范围内时，不会发生饱和。舍入前的有效因子为 $2^{\ell-r-\delta}$；固定归一化已吸收到通道缩放因子中。

同一有效移位也可以反向索引：当选择器的取值范围允许时，非负扩展选择器s 对应于 $r = \ell - s$。只有舍入和范围处理也一致时，任一约定指定的重构才相同。截断会在进行足够宽的中间计算后丢弃低位；而保留溢出值的低字节则会导致回绕。

附录 A 给出了一个 SQ 重构的实例。

3.2.4 存储与算术

对于完整分组，编码存储为

$$B_{\mathrm{SQ}} = x + \frac{m+n}{y} \quad \text{bits per weight.} \tag{7}$$

通道缩放因子、可选偏移量、变换符号和填充是额外信息。有符号性、移位解释和范围处理通过指定方程（6）来完善描述符。

SQ2–SQ6 主要用于重构 INT8 操作数。分块实现会将编码解包并移位到计算片段或有界暂存缓冲区中，然后执行 INT8 矩阵乘加，并使用 INT32 累加。由于 P 已包含局部自适应，因此沿 K 进行点积时无需浮点分组尺度。由此产生的效率取决于解码吞吐量、暂存流量以及与收缩运算的融合，而这些因素均纳入计时。二次幂缩放和共享指数是成熟的技术 [28]；SQ 将它们与共享位追加和位宽缩减相结合，旨在适配常规整数操作数。

3.3 SHI 四位乘积与分组校正

SQ 会在乘法前重构整数操作数。对于某些低位格式，即使存储编码不超过四位，该操作数仍需要 INT8。我们开发了移位整数（SHI），作为 SQ 的一种变体，旨在使用原生 INT4 算术。当重构权重可用 INT4 表示时，可直接使用；否则，SHI 会通过分别计算 INT4 乘积来处理低位基值和共享分组校正。暂且忽略通道缩放因子，这种表示可写为

$$w_{j,k} = \rho q_{j,k} + r_{j,g(k)}, \tag{8}$$

其中， q 是单个权重的基值操作数， r 是其所在组共享的有符号校正，而ρ 则设定这两部分的相对大小。操作数采用对存储字段的平衡表示；具体的转换方式和任何截断操作都属于数值配置的一部分。SHI2–SHI4 使用 INT2 至 INT4 基值编码，因此其基值操作数适合使用有符号 INT4 表示，并可采用 INT4 算术。五位或更宽的基值则需要更宽的操作数（例如 INT8），也无法保留这一优势。例如，SHI4 G8 为每个权重使用一个四位基值，并为每八个权重使用一个四位校正。这些字段在加入缩放因子和其他元数据之前，占用每个权重 $4 + 4/8 = 4.5$ 位。

与其他分组格式一样，SHI 会使用额外位数来改善每个组的表示。NVFP4 每 16 个四位值存储一个八位尺度，因此在加入全局尺度和填充之前，每个值需要 4.5 位；MXFP4 每 32 个值存储一个八位尺度，因此每个值需要 4.25 位 [24, 25]。当权重和激活值都采用这些格式时，其有效速率分别为 W4.5A4.5 和 W4.25A4.25。在支持原生分块缩放 FP4 指令的硬件上，缩放因子会在矩阵运算中应用，使这些格式能够使用原生 FP4 矩阵吞吐量。相比之下，SHI 通过整数乘积计算其分组校正。

The 以下 SQ 字段结构展示了 SHI 如何保留四位算术 :

- **SHI2：SQ2-G4A1S1**。扩展并移位后的权重可容纳在一个有符号 INT4 操作数中。因此，可以在内核中重构该权重，并直接与 INT4 激活值相乘

得到一个乘积。由于激活值的位数多于二位基码，我们主要采用单乘积路径，无需单独进行激活值校正。

- **SHI3：SQ3-G8A3S1**。三位基码和一位移位可组成四位主操作数；在选定的重构和范围处理方式下，三个附加位和移位可组成四位分组校正操作数。我们使用两个乘积来计算这些权重分量。四位激活值的位数多于基础编码，因此我们主要采用双乘积路径，不进行激活值校正。

- **SHI4：SQ4-G8A4S0**。由于没有移位选择器，四位基值和四位共享追加字段构成主权重分量和分组权重分量；平衡表示将它们映射为有符号操作数。由于激活值也采用 INT4，我们使用相应的 SQ4-G8A4S0 风格激活值表示，并通过第三个乘积纳入其分组校正。

对于 SHI4 G8，我们分别将仅基码、权重校正以及权重和激活值校正计算称为 W4A4、W4.5A4 和 W4.5A4.5。分数位数描述的是在八个值上取平均的共享四位字段；它们不会改变 INT4 操作数宽度。相同的乘积结构也适用于其他 SQ 格式，其有效位数可能有所不同。

对于 SHI2，单乘积路径使用完整的重构 INT4 权重。对于拆分为基码和校正项的表示，下面的仅基码表达式展示了省略校正项的影响。权重和激活值校正方程使用公式（8）中的未饱和表示；截断组合 INT8 操作数会改变截断项的值。

3.3.1 主乘积（W4A4）

最简单的方案是将权重基码与激活值的四位近似相乘。设$\boldsymbol{\xi}$ 为经过旋转和输入通道缩放后的输入行，并将每个激活值近似为$\xi_k \simeq s a_k$，其中 a_k 是四位码， s 是其缩放因子。仅使用权重基码可得

$$\widehat{y}_j^{\text{base}} = c_{o,j}\rho s \sum_k q_{j,k} a_k. \tag{9}$$

这是跨所有 K 个输入通道的一个 INT4 乘积。该方案成本最低，但会忽略存储的权重校正。

3.3.2 分组权重校正（W4.5A4）

权重校正路径会加上共享权重校正。由于组内每个权重共享同一校正项，我们可以先对该组中的激活值求和，再将总和乘以校正项：

$$\sum_k (\rho q_{j,k} + r_{j,g(k)})\xi_k = \rho \sum_k q_{j,k}\xi_k + \sum_g r_{j,g} \sum_{k\in g} \xi_k. \tag{10}$$

第一个乘积仍涵盖所有 K 权重。校正乘积每组只需一个值，因此其长度为 K/y：对于包含八个元素的组，其长度是主乘积的八分之一。

要在 INT4 中运行校正乘积，我们将激活码求和， $h_g = \sum_{k \in g} a_k$，并将缩放后的和量化为 $sh_g \simeq tb_g$。其中， b_g 是一个四位码， t 是每令牌缩放因子。两个乘积计算

$$\widehat{y}_j^{\text{weight}} = c_{o,j} \left[\rho s \sum_k q_{j,k} a_k + t \sum_g r_{j,g} b_g \right]. \tag{11}$$

第二个乘积近似表示权重校正的贡献。组和可能超出 INT4 范围，因此对其进行量化会引入误差。激活值本身仍仅使用其基准四位近似。

3.3.3 分组激活校正（W4.5A4.5）

激活校正路径会为每个激活组添加一项共享的校正 p:

$$\xi_k \simeq s(\tau a_k + c_{g(k)}), \tag{12}$$

这里， a 和 c 是四位码，而 τ 设定了它们的相对大小。为高效计算校正，定义每组激活基值之和以及重构权重之和:

$$S_{a,g} = \sum_{k \in g} a_k, \qquad S_{w,j,g} = \rho \sum_{k \in g} q_{j,k} + y r_{j,g}. \tag{13}$$

T权重表示与激活值表示的乘积可写为 as

$$y_j \simeq c_{o,j} s \left[\rho\tau \sum_k q_{j,k} a_k + \tau \sum_g r_{j,g} S_{a,g} + \sum_g S_{w,j,g} c_g \right]. \tag{14}$$

第一项是主乘积，第二项计入权重校正，第三项计入激活值校正。权重和包含 yr，因此第三项也计入两项校正之间的交互作用；无需第四个乘积。

组和可能 比 INT4 更宽，因此我们在进行两次校正乘积之前先对其进行量化 :

$$s\tau S_{a,g} \simeq t b_g, \qquad S_{w,j,g} \simeq u_j d_{j,g}, \tag{15}$$

其中， b 和 d 是四位编码， t 是每令牌缩放因子， u_j 是按输出行拟合的缩放因子。执行的三乘积近似为

$$\widehat{y}_j^{\text{weight+activation}} = c_{o,j} \left[\rho\tau s \sum_k q_{j,k} a_k + t \sum_g r_{j,g} b_g + s u_j \sum_g d_{j,g} c_g \right]. \tag{16}$$

在评估的 SHI4 构造中， $\rho = \tau = 16$ 且三个乘积都使用四位操作数。对组和进行量化会引入额外误差，因此，公式 (16) 中执行的结果是对公式 (14) 中展开式的近似。

配置文件	评估信息	归约长度	G8 处的运算	G16 处的运算
一个乘积	基础权重和激活	K	$1.000K$	$1.000K$
两个乘积	添加分组权重	$K + K/y$	$1.125K$	$1.0625K$
三个乘积	校正 同时校正激活值， 包括交叉项	$K + 2K/y$	$1.250K$	$1.125K$

表 1：每个输出的逻辑收缩运算量，其中 K 是 W4A4 主乘积的运算量。

SHI 旨在利用矩阵乘法中量化权重的结构，在可以使用 INT4 乘积时避免完整长度的 INT8 乘积。以吞吐量归一化进行比较，若 INT4 的算术吞吐量是 INT8 的两倍，则长度为 K 的 INT8 乘积相当于$2K$ 的 INT4 运算量。表 1 仅统计 INT4 收缩运算，不包括准备或数据搬运。对于大小为八的组，两乘积和三乘积路径分别需要 $1.125K$ 和 $1.25K$ 的运算量；组大小为十六时，则分别降至 $1.0625K$ 和 $1.125K$。较大的组可以减少校正运算，但可能影响量化精度。运行时还取决于激活值的准备与打包、组和的计算、数据搬运以及输出转换。多个乘积可以融合到一个内核中，因此乘积数量不一定等于启动次数。

预先准备权重和分量 d 可能会增加驻留内存。如果四位 q、 r 和 d 全部保留，则每个权重需要 $4+8/y$逻辑编码位，不包括缩放因子和填充。对于 G8，序列化基项和校正占每个权重 4.5 位，而这种预先准备的三平面表示占 5.0 位。如果保留派生分量，则在驻留内存统计中计入该分量，即使它是在加载时生成的。

激活量化和数值配置的详细信息见附录 B。

4 投影层级评估

我们首先单独考察一个 MLP 投影，以测量每种量化路径的准确率和延迟。

4.1 基准测试协议

我们在 RTX 5090 上单独测试 Qwen3.8-27B 的第 31 层 MLP 下投影（$K = 17,$ 408， $N = 5,$ 120）。所有格式都使用未加权平方重构误差，拟合到相同的 BF16 权重。我们以相对弗罗贝尼乌斯范数衡量权重误差，并以每个 WikiText-2 上下文的 255 个激活向量上的平均相对 L_2 误差衡量输出误差。测量输出误差时，我们将残差设为零，并将输出舍入为 BF16。

我们在 4,096 个 token 下对每个投影计时。进行冷缓存计时时，llama.cpp 以驻留在内存中的 FP32 输入和就绪的 FP32 残差开始，并写出 FP32 输出。Shifter 和浮点 CUDA 路径使用 BF16 输入、残差和输出。计时区间包括激活预处理、矩阵乘法、缩放、残差加法、后端转换、主机端调度和同步。表格报告中位延迟，并以 llama.cpp Q2_K 和 Q4_K 作为

相应的吞吐量基线。Q2_K 也作为完整检查点比较中的参考格式（表5）。每权重打包比特数（每权重位数）统计权重编码和组字段。Shifter 还存储通道缩放因子和 H512 符号。llama.cpp 格式作为软件基线 [9]。

4.2 两比特投影结果

权重／引擎	激活	打包 bpw	权重误差	输出误差	4,096 个 token 的延迟	加速比 与 Q2_K 相比
Q2_0 (G32) / llama.cpp	Q8_1	2.500	33.574%	30.680%	2,521.51 µs	1.65×
Q2_K / llama.cpp	Q8_1	2.625	26.510%	24.152%	4,162.47 µs	1.00×
SQ2-G4A1S1 / Shifter（SQ2）	INT8	2.500	**25.281%**	**22.971%**	1,504.74 µs	2.77×
SQ2-G4A1S1 / Shifter (SHI2)	INT4	2.500	**25.281%**	27.924%	**914.34 µs**	**4.55**×

表2：两位投影误差与延迟。加速比以 llama.cpp Q2_K 基线为准。

我们在五个 WikiText-2 测试上下文上测量输出误差。SQ2 和 SHI2 共用一组 SQ2-G4A1S1 权重拟合结果。SQ2 使用 INT8 激活值，SHI2 则使用第3.3节所述的单乘积 W4A4 路径。自定义的 Q2_0 比较格式每个 FP16 缩放因子对应 32 个权重，打包每权重比特数为 2.500。

SQ2-G4A1S1/INT8 的打包每权重比特数为 2.500，而 Q2_K 为 2.625。在 Q2_K 吞吐量下，其输出误差从 24.152% 降至 22.971%。使用相同的拟合权重时，单乘积 INT4 路径将延迟从 1,504.74µs 降至 914.34µs，但输出误差从 22.971% 升至 27.924%。

4.3 四位投影结果

权重／引擎	激活	打包 bpw	权重误差	输出误差	4,096 个 token 的延迟	加速比 与 Q4_K 相比
Q4_0 / llama.cpp	Q8_1	4.500	8.246%	7.496%	2,616.33 µs	1.10×
Q4_K / llama.cpp	Q8_1	4.500	**7.124%**	**6.688%**	2,873.24 µs	1.00×
SQ4-G8A2S2 / Shifter	INT8	4.500	7.457%	6.804%	1,481.26 µs	1.94×
SHI4 (G8A4S0) / Shifter	INT8	4.500	8.632%	7.841%	1,490.18 µs	1.93×
	SHI4	4.500	8.632%	12.236%	1,014.05 µs	2.83×
NVFP4 / CUDA	FP8	4.500	8.178%	7.837%	1,472.14 µs	1.95×
	NVFP4	4.500	8.178%	10.993%	767.71 µs	3.74×
MXFP4（E2M1 + E8M0，G16）/ CUDA	FP8	4.500	11.007%	10.889%	1,473.27 µs	1.95×
	NVFP4	4.500	11.007%	12.856%	**753.82 µs**	**3.81**×

表 3：四位投影误差和延迟。加速比使用 Q4_K 基线。SHI4 激活行使用校准的静态截断。

每两行 SHI4、NVFP4 或 MXFP4 共用一个拟合权重张量。Shifter 路径使用 H512，而 llama.cpp 和浮点路径使用原始基。SHI4 激活行使用 Shifter Engine 原生的三乘积下投影。我们使用第一个捕获的上下文，校准一个固定的张量范围 NVFP4 激活缩放因子和两个固定的 SHI4 裁剪比例。对于

对于 SHI4，我们分别为基础激活值和 G8 基码和搜索五个因子。NVFP4 在线计算其 G16 缩放因子。我们在剩余四个上下文中测量所有格式的输出误差。

llama.cpp CUDA 图从常驻的 FP32 输入中准备 Q8_1 激活值。FP8 计时包括每次调用时将打包权重扩展为 FP8，并采用经过调优的 cuBLASLt 调度方案。MXFP4 变体每 16 个权重使用一个 E8M0 缩放因子，打包后为 4.5 每权重位数。

在此投影上，采用四位激活值的 SHI4 速度是 Q4_K 的 $2.83\times$倍，输出误差为 12.236%。
SQ4-G8A2S2/INT8 的速度是 Q4_K 的 $1.94\times$ 倍，输出误差为 6.804%；相比之下，Q4_K 的输出误差为 6.688%。

5 全模型评估

接下来，我们评估完整检查点，以衡量投影层面的结果能否延伸至模型质量和预填充。我们分别比较二位、四位和八位格式，并使用相应的 BF16 模型作为质量参考。

5.1 WikiText-2 测试方案

我们采用 WikiText-2 [18]，因为广泛使用的本地推理引擎 llama.cpp 提供了一套标准流程，用于将量化模型与其高精度参考模型进行比较。其评估会报告 KL 散度、top-1 一致率和困惑度；这些指标也用于社区量化比较 [10, 30]。

每个模型均使用 llama.cpp 的概率和 KL 散度实现，根据来自 580 个 WikiText-2 测试上下文的 147,900 个下一个词元目标进行评分。每个检查点都以相应的原始 BF16 模型作为参考。检查点名称标明了其在混合精度模型中采用的主要量化格式。

KL 散度是我们的主要质量指标，因为它衡量了相对于 BF16 参考模型，量化对模型下一个词元概率分布的改变程度。top-1 一致率表示量化模型与 BF16 模型预测出相同最高概率词元的位置所占比例。困惑度（PPL）衡量模型在测试文本上的预测表现。

我们评估未经校准的仅权重量化拟合，并与已发布检查点进行比较。若能获取提供方的校准数据，我们就用它来拟合相应的 Shifter 检查点。

5.2 二比特检查点

5.2.1 仅权重拟合

仅权重模型	KL ↓	Top-1 ↑	困惑度 ↓
Shifter-SQ2	**0.20001**	**81.6633%**	7.91754
Shifter-SHI2	0.21210	80.9838%	**7.89657**
Q2_K 配方（局部重拟合）	0.27849	78.2596%	8.58626

表 4：仅权重量化拟合的完整检查点 WikiText-2 质量。

我们在相同的上下文和目标上评估仅权重量化拟合。Shifter 运行使用 INT8 激活值，并禁用离群值保护。Q2_K 对比项在未经校准的情况下，重新拟合 Bartowski 的混合精度方案。

表 4 显示，与 Q2_K 重新拟合相比，SQ2 和 SHI2 分别将 KL 散度降低了 28.2% 和 23.8%。两者还提升了 top-1 一致率并降低了困惑度。

5.2.2 已发布检查点对比

模型	KL ↓	Top-1 ↑	困惑度 ↓	4,096 个令牌预填充 (令牌/秒) ↑
Shifter-SQ2	**0.15322**	**84.0237%**	**7.57241**	8,378.4 (2.71×)
Shifter-SHI2	0.15537	83.8316%	7.58638	**11,662.4 (3.77×)**
Bartowski Q2_K	0.16627	83.3110%	7.70266	3,090.7 (1.00×)

表5：完整检查点质量和预填充吞吐量。加速比的计算方式是将各未舍入的预填充速率除以 Bartowski 发布的 Q2_K 的速率。

我们将 Qwen3.8-27B 的 Shifter-SQ2 和 Shifter-SHI2 与 Bartowski 在 llama.cpp 中发布的 Q2_K 检查点进行比较。表5将 WikiText-2 质量与 4,096 个令牌预填充吞吐量并列展示。

两个 Shifter 检查点在质量评估中都使用 INT8 激活值。质量评估使用 llama.cpp 的概率辅助函数，对相同的上下文和目标进行评估。预填充时，Shifter-SQ2 使用 INT8 激活值，Shifter-SHI2 使用四位激活值。每次预填充运行都使用单个 4,096 令牌块、BF16 键值缓存和全新的提示状态。我们报告两次预热后进行的三次运行的中位数。

在这项已发布检查点对比中，SQ2 的 KL 散度和困惑度最低，top-1 一致率最高。与 Q2_K 相比，SQ2/INT8 在第31层下投影上的吞吐量达到 $2.77\times$ （表2），在全模型预填充期间达到 $2.71\times$ 。采用四位激活值的 SHI2 在投影上的吞吐量达到$4.55\times$ ，在预填充期间达到 $3.77\times$ （11,662.4 令牌/秒）。两项测量均使用 4,096 个令牌。投影使用本地拟合的 Q2_K 张量，而预填充使用 Bartowski 发布的检查点。

5.3 四位检查点配对

模型	KL ↓	Top-1 ↑	困惑度 ↓
Qwen3.8-27B SHI4（A16，受保护）/ Shifter	**0.01373**	**94.9824%**	**6.94277**
Qwen3.8-27B NVFP4（BF16）/ NInfer	0.01875	93.9249%	7.04771
Qwen3.8-27B SQ4 / Shifter	**0.01970**	**93.9108%**	**6.99283**
Qwen3.8-27B 分组 INT / NInfer	0.03256	92.1400%	7.14469
Qwen3.6-35B-A3B SQ4 / Shifter	**0.01828**	**94.2042%**	**6.75563**
Qwen3.6-35B-A3B 分组 INT / NInfer	0.02400	93.3529%	6.79117

表 6：完整检查点 WikiText-2 质量。粗体表示每对中的较优值。

对于 Qwen3.8-27B，我们将 SHI4 与 NVFP4 对比，将 SQ4 与分组 INT 对比。对于 Qwen3.6-35B-A3B，我们将 SQ4 与分组 INT 对比。六次运行均使用 BF16 键值缓存。

SQ4 检查点在其 SQ4 投影上使用 INT8 激活算术。SHI4 质量评估使用经 MACaQ 校准的 GGUF，在平衡 MLP 上使用 A16 激活，并在符合条件的辅助 INT8 投影上启用离群值保护。NInfer 在 NVFP4 投影上使用 BF16 激活。Shifter 和 NInfer 的质量评估使用来自相应参考模型独立运行的 BF16 参考 logits。

SHI4 的 KL 散度和困惑度低于 NVFP4，top-1 一致率则更高。两个 SQ4 检查点在这三项指标上均优于其分组 INT 基线。

另一项静态四比特 SHI4 评估衡量了辅助 INT8 保护的影响：KL 散度从 0.05569 降至 0.03531，top-1 一致率从 89.9094% 提升至 92.0230%。

5.4 八比特检查点比较

运行时	KL ↓	Top-1 ↑	困惑度 ↓	4,096 个令牌预填充（令牌/秒）↑
vLLM FP8 W8A8	0.00774	96.0297%	**6.96035**	7,226.8
SQ8 / Shifter	**0.00211**	**98.0439%**	6.96039	**9,206.3**

表 7：八位全模型质量与单批次预填充。预填充使用单个 4,096 令牌块、初始状态和一个贪心下一个令牌结果。速率为两次预热后进行三次测量所得的中位数。

在八位精度方面，我们将 Shifter Engine 的 SQ8 检查点与 vLLM 在 Qwen3.8-27B [12, 31] 上使用的在线 FP8 W8A8 进行比较。Shifter 在带符号哈达玛旋转后拟合逐输出通道 INT8 权重，并使用固定的逐令牌 INT8 激活缩放因子。vLLM 在加载时将 BF16 检查点量化为逐张量 E4M3 权重，并通过 CUTLASS 使用动态缩放的 FP8 激活。两者均使用 BF16 键值缓存和相同的 BF16 WikiText-2 参考配置，并在计时过程中禁用 CUDA 图、前缀复用和推测解码。

Shifter 的预填充速度提升了 27.4%，KL 散度降低了 72.7%，top-1 一致率提高了 2.01 个百分点。两者的困惑度值几乎相同。

6 四位服务评估

我们将 Shifter Engine 与 NInfer 进行比较。NInfer 是由社区开发的本地推理引擎，专为部分 Qwen 模型打造 [20]。此外，也有针对其他 GPU 和部署环境的分支版本受到 NInfer 的启发 [1]。凭借其专业化设计和已公布的性能数据，NInfer 成为这些模型强有力的社区基线。我们评估三组四位模型与格式组合在单请求预填充和结构化解码中的表现，随后评估并发解码。

6.1 部署配置

服务对比采用与表 6 相同的三组模型和格式组合。在服务评估中，Shifter 使用经 MACaQ 校准的 SHI4 工件，采用静态裁剪的四位 MLP 激活值，并关闭辅助 INT8 投影的离群值保护。两种引擎都采用依赖形状的激活调度。

RTX 5090 上的 Shifter 测量使用 Linux，功率上限为 450 W，驱动程序版本为 595.71.05，CUDA 版本为 13.2，PyTorch 版本为 2.9.1+cu128。Shifter 启用短批次 MoE 专家权重复用。服务表格采用 NInfer 针对修订版 `a754fcf3` 公布的速率。Shifter 在注意力Q/K/V 上使用归一化H256，并采用 INT8 KV，每个头向量配有一个 FP16 缩放因子。NInfer 对 27B 使用 FP8 E4M3 G128 KV，对 35B 使用 G64 INT8 KV。

6.2 单请求预填充与结构化解码

模型与对比	预填充 7.7K	预填充 260K	结构化 MTP3
Qwen3.8-27B SHI4 / Shifter	10,252.3	3,550.0	212.1±24.0
Qwen3.8-27B NVFP4 / NInfer	**12,819.1**	**4,016.4**	**231.7**
Shifter / NInfer	0.80×	0.88×	0.92×
Qwen3.8-27B SQ4 / Shifter	**8,221.9**	**3,055.2**	**242.6±16.6**
Qwen3.8-27B 分组 INT / NInfer	3,331.9	2,139.4	214.7
Shifter / NInfer	2.47×	1.43×	1.13×
Qwen3.6-35B-A3B SQ4 / Shifter	**32,039.0**	**8,738.0**	**784.0±37.6**
Qwen3.6-35B-A3B 分组 INT / NInfer	17,705.4	5,247.0	779.6
Shifter / NInfer	1.81×	1.67×	1.01×

表 8：单请求服务吞吐量（令牌/秒）。对于每个 27B 检查点和 35B 检查点，Shifter 结构化 MTP3 分别报告三次请求的均值± 样本标准差，其中 27B 检查点为 15 次请求。NInfer 结构化 MTP3 的数值为均值。比率采用未四舍五入的速率计算。粗体表示每对结果中较高的实测吞吐量。

对于预填充，我们使用 NInfer 的 7,680 和 260,096 令牌针式检索测试集，并禁用推测执行。对于使用三令牌多令牌预测（MTP3）和优化提议头 [11, 13] 的结构化解码，我们使用其 JSONL、CSV 和 SQL 测试集。Shifter 启用 CUDA 图，短预填充使用 4,096 令牌的分块，长预填充和解码使用 1,024 令牌的分块，并同时禁用前缀复用和思考模式。采样使用温度 0.6、top-p 0.95、top-k 20、最小概率 0、存在惩罚 1 和频率惩罚 0。结构化 MTP3 为每个请求设置 4,096 令牌的输出上限。每个模型在每轮阶段扫描前准备一次，每个请求均使用全新的序列状态。

表 8 将输入令牌数除以服务器预填充阶段时间，并对短提示词的前 3 个固定种子取平均值。极端提示词使用第 1 个固定种子。Shifter 使用针对各工作负载调优的配置文件，并在预填充阶段生成一个输出令牌。我们将每个请求首个输出令牌之后的输出令牌数除以记录的解码时间，计算其解码速率。对于 27B，Shifter 每个测试样例使用一个随机种子；对于 35B，每个测试样例使用五个随机种子。我们保留实际终止长度和完成长度，并排除加载和预热时间。

相较于分组 INT，SQ4 在 27B 上的短提示词和极端提示词预填充吞吐量分别达到 $2.47\times$ 和 $1.43\times$，在 35B 上分别达到 $1.81\times$ 和 $1.67\times$。在结构化解码方面，35B 的平均吞吐量为：Shifter 每秒 784.0 个令牌，NInfer 每秒 779.6 个令牌。

6.3 并发 MTP3 解码

模型与对比	$C=1$	$C=2$	$C=4$	$C=8$
Qwen3.8-27B SHI4 / Shifter	144.4 / 46.8%	276.6 / 46.5%	503.4 / 47.6%	809.5 / 46.0%
Qwen3.8-27B NVFP4 / NInfer	**147.7**/ 46.2%	**291.0**/ 48.7%	**522.2**/ 45.8%	**922.4** /46.1%
Qwen3.8-27B SQ4 / Shifter	**156.9**/ 46.1%	**283.6**/ 45.8%	**452.3**/ 47.1%	**765.5**/ 46.7%
Qwen3.8-27B 分组 INT / NInfer	136.5 / 44.4%	253.3 / 45.2%	398.1 / 46.1%	582.4 / 46.4%
Qwen3.6-35B-A3B SQ4 / Shifter	**654.3** / 68.3%	**1,004.5** / 69.0%	**1,237.2**/ 69.4%	**1,721.5** / 68.8%
Qwen3.6-35B-A3B 分组 INT / NInfer	642.5 / 68.6%	907.2 / 66.3%	1,213.5 / 69.6%	1,380.7 / 68.0%

表 9：并发 MTP3 解码。每个单元格先报告稳态吞吐量（令牌/秒），再报告全波接受率。粗体表示每组中实测吞吐量较高的一项。

对于每种并发度 $C \in \{1,2,4,8\}$，我们使用 NInfer 的 `long_decode_aime26_15` fixture 为每个序列发送一个请求。各请求使用不同的固定种子，启用思考模式，输出预算为 8,192 个令牌，上下文限制为 16,384 个令牌。CUDA 图、MTP3、提议头、采样和预填充块大小均沿用单请求设置。请求可以在遇到 EOS 时结束。Shifter 基准测试会为兼容的波次重复使用同一个预部署实例。对于 27B SQ4，它会将分块权重固定在最多八个验证器列上；对于 27B SHI4，则会将分块门控/上投影与下投影绑定固定在最多十六个验证器列上。更宽的波次和 35B SQ4 使用行主序权重。结构化解码使用相同的窄布局选择。

表 9 统计的是完整的一秒时间段内已提交的解码令牌；在这些时间段内，所有 C 序列均已解码就绪，每个实际解码批次的大小均为 C，且没有发生预填充或恢复。这不包括升 ramp、排空和未填满区间。接受率是完整请求波次中接受令牌数与草拟令牌数之比。每个单元格代表一个波次。

在并发度为八时，Shifter的吞吐量分别为 NInfer 在 27B SHI4、27B SQ4 和 35B SQ4 上的 $0.88\times$、$1.31\times$ 和 $1.25\times$ 。

6.4 跨 GPU 吞吐量缩放

我们在涵盖 Ampere、Ada 和 Blackwell 的四款 GPU 上评估了两个 SQ4 检查点。图 1 将测得的预填充速率与峰值密集 INT8 张量吞吐量相关联，并将结构化 MTP3 解码速率与峰值设备内存带宽相关联 [21, 22]。在四款 GPU 上，线性拟合结果显示，预填充和解码的拟合优度分别为$R^2 = 0.987$–0.992 和 $R^2 = 0.939$–0.970。例如，35B-A3B SQ4 检查点在两款 Ampere GPU 上的预填充吞吐量均超过 10,800 令牌/秒，结构化解码吞吐量均超过 330 令牌/秒。表 10 给出了测得的速率。

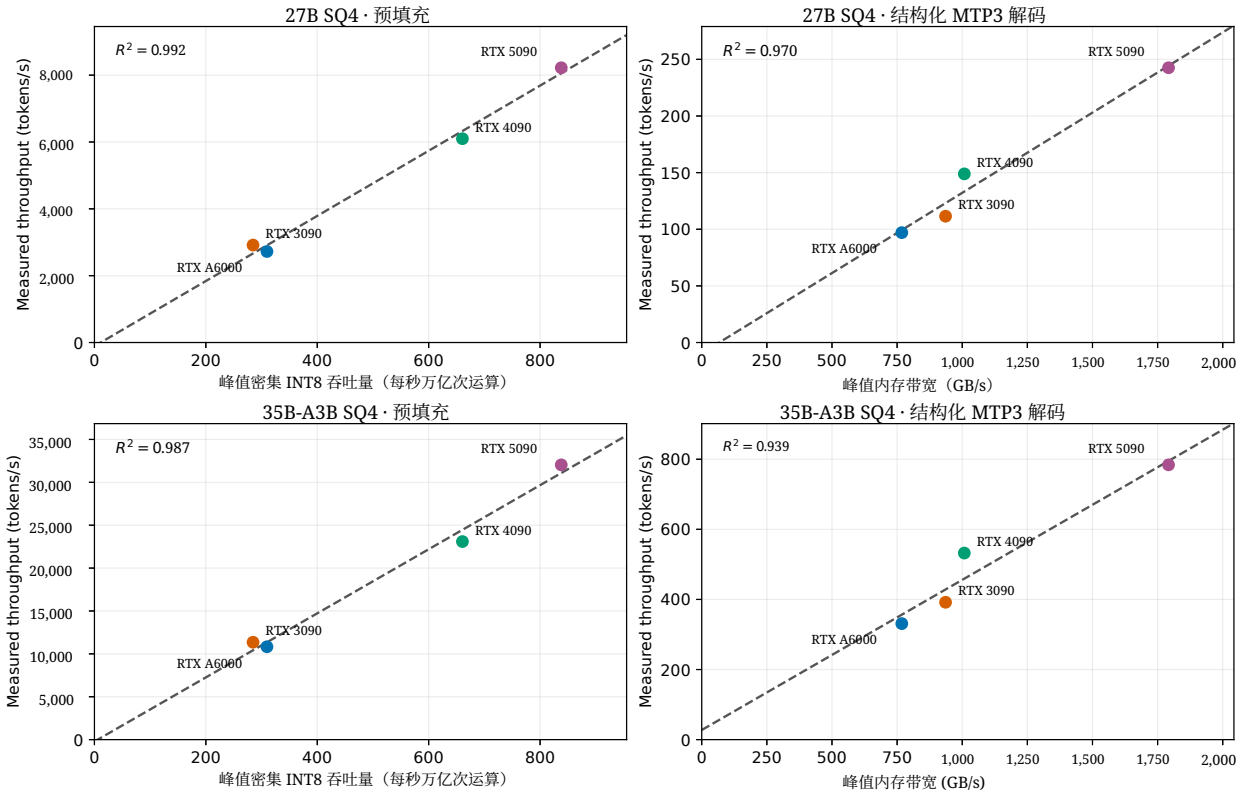


图 1：四款 GPU 上、针对 7,680 令牌预填充和单请求结构化 MTP3 解码的 SQ4 吞吐量。虚线表示截距自由的最小二乘拟合，并延伸至测量点之外。每个 R^2 描述四个观测值。不同部署中的预填充块大小、上下文分配和种子数量各不相同。

7 结论

我们提出了移位量化（SQ）和移位整数（SHI），用于通过传统整数矩阵运算实现低比特推理。SQ 采用共享组信息存储紧凑权重编码，并通过位运算重构整数操作数。这使局部自适应发生在

操作数中，而无需在矩阵归约过程中使用浮点分组尺度。SHI 基于这一表示，采用四位乘积，并在需要时通过更短的整数乘积计算分组权重与激活校正。两种方法都保留紧凑权重，同时允许主乘积在整个归约维度上累加。这种结构与硬件加速整数矩阵指令相契合，有助于将低位存储转化为更快的推理。

在仅权重二位比较中，相较于未经过校准、重新拟合的 Q2_K 配方，SQ2 和 SHI2 分别将 WikiText-2 KL散度降低了 28.2% 和 23.8%。SQ 和 SHI 可以为较小的组分配移位或校正，而无需在主归约过程中为每个组引入浮点缩放因子。这种更细的粒度赋予表示更大的局部灵活性，并可能有助于提升观察到的质量。SQ2 在经测试的投影上还以更低的输出误差实现了 $2.77\times$ Q2_K 的吞吐量。与已发布的 Q2_K 检查点相比，SQ2 和 SHI2 分别达到 $2.71\times$ 和$3.77\times$的 4,096 个令牌预填充吞吐量。

与经测试的分组整数检查点相比，SQ4 提升了质量，并在两种 Qwen 模型和不同提示长度下，实现了其 NInfer预填充吞吐量的 1.43–$2.47\times$。在并发度为八时，对于 27B 和 35B 模型，SQ4 的解码吞吐量分别达到对应 NInfer 检查点的 $1.31\times$ 和 $1.25\times$。在八位精度下，与在线 FP8 相比，SQ8 将 WikiText-2 KL散度降低了 72.7%，并将批量为一的预填充吞吐量提升了 27.4%。

四位激活量化和分组校正会引入额外的近似误差，其准备过程和数据搬运也会增加运行时开销。在测试的服务配置中，SHI4 仍然比原生NVFP4 更慢。

在涵盖 Ampere、Ada 和 Blackwell 的四款 GPU 上，SQ4预填充吞吐量与稠密INT8吞吐量相当，而结构化 MTP3 解码则受内存带宽影响。

参考文献

[1] Alphastorm贡献者。支持RTX 4090和3090的NInfer分支，未注明日期。URL `https://github.com/alphastorm/ninfer`。[2] AMD。AMD CDNA 4架构白皮书，2025年。URL `https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-4-architecture-whitepaper.pdf`。[3] S. Ashkboos等。QuaRot：旋转大语言模型中的无离群值四位推理。在*NeurIPS*，2024年。[4] S. Dai等。VS-Quant：用于精确低精度神经网络推理的逐向量缩放量化。在*MLSys*，2021年。[5]DeepSeek AI。DeepSeek-V4-Pro模型卡，2026年。URL `https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro`。[6] T. Dettmers、M. Lewis、Y. Belkada和L. Zettlemoyer。LLM.int8()：面向大规模Transformer的八位矩阵乘法。在*NeurIPS*，2022年。

[7] T. Dettmers、A. Pagnoni、A. Holtzman和L. Zettlemoyer。QLoRA：量化大语言模型的高效微调。在*NeurIPS*，2023年。[8] E. Frantar、S. Ashkboos、T. Hoefler和D. Alistarh。GPTQ：生成式预训练变换器的精确训练后量化。在 *ICLR*，2023年。[9] ggml-org。llama.cpp：张量编码方案，未注明日期。URL https://github.com/ggml-org/llama.cpp/wiki/Tensor-Encoding-Schemes。[10] ggml-org。llama.cpp：困惑度和KL 散度评估，未注明日期。URL https://github.com/ggml-org/llama.cpp/blob/master/tools/perplexity/README.md。[11] F. Gloeckle、B. Y. Idrissi、B. Rozière、D. Lopez-Paz和G. Synnaeve。通过多令牌预测实现更优、更快的大语言模型。在 *ICML*，第15706–15734页，2024年。[12] W. Kwon等。借助PagedAttention实现大语言模型服务的高效内存管理。在*SOSP*，2023年。[13] Y. Leviathan、M. Kalman和Y. Matias。通过推测解码实现Transformer的快速推理。在*ICML*，第19274–19286页，2023年。[14] J. Lin等。AWQ：面向端侧大语言模型压缩和加速的激活感知权重量化。在 *MLSys*，2024年。[15] Y. Lin等。QServe：面向高效大语言模型服务的W4A8KV4量化与系统协同设计。在*MLSys*，2025年。[16] Z. Liu、J. Yuan、H. Jin、S. Zhong、Z. Xu、V. Braverman、B. Chen和X. Hu。KIVI：无需调优的KV缓存非对称2位量化。在 *ICML*，第32332–32344页，2024年。[17] 美团。LongCat-2.0模型卡，2026年。URL https://huggingface.co/meituan-longcat/LongCat-2.0。[18] S. Merity、C. Xiong、J. Bradbury和R. Socher。指针哨兵混合模型。在 *ICLR*，2017年。[19] 月之暗面AI。Kimi K3模型卡，2026年。URL https://huggingface.co/moonshotai/Kimi-K3。[20]NInfer贡献者。NInfer：服务实现与性能方法，未注明日期。URL https://github.com/Neroued/ninfer/tree/a754fcf3bf524494aa927c2f54e0cbfb58f569ab。修订版 a754fcf3。[21] 英伟达。NVIDIA Ampere GA102 GPU架构，未注明日期。URL https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf。版本2.1。[22] 英伟达。NVIDIA RTX Blackwell GPU架构，未注明日期。URL https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf。

[23] 英伟达。并行线程执行指令集架构，矩阵乘累加的块缩放，未注明日期。URL https://docs.nvidia.com/cuda/parallel-thread-execution/index.html。[24]英伟达等。《使用 NVFP4 预训练大语言模型》。*arXiv* 预印本 *arXiv:2509.25149*，2025年。[25]开放计算项目。《OCP 微缩放格式（MX）规范》，2023年。URL https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf。版本 1.0。[26] A. V. Oppenheim。使用块浮点运算实现数字滤波器。*IEEE 音频与电声学汇刊*，18(2):130–136，1970年。[27] Qwen 团队。Qwen 3.8 发布公告，2026年。URL https://qwen.ai/blog?id=qwen3.8。[28] B. Rouhani 等。共享微指数：小小移位，成效显著。载于*ISCA*，2023年。[29]B. D. Rouhani 等。用于深度学习的微缩放数据格式。*arXiv* 预印本 *arXiv:2310.10537*，2023年。[30] Unsloth AI。Unsloth Dynamic v2.0 GGUF，未注明日期。URL https://www.unsloth.ai/blog/dynamic-v2。[31] vLLM 贡献者。在线量化，未注明日期。URL https://docs.vllm.ai/en/stable/features/quantization/online/。[32] G. Xiao、J. Lin、M. Seznec、H. Wu、J. Demouth 和 S. Han。SmoothQuant：面向大语言模型的精准高效训练后量化。载于*ICML*，2023年。[33] Z. Yao 等。ZeroQuant：面向大规模 Transformer 的高效且经济的训练后量化。载于*NeurIPS*，2022年。[34] Y. Zhao 等。Atom：用于高效精准大语言模型服务的低比特量化。载于*MLSys*，2024年。

SQ 工作示例

考虑四个经过变换和通道缩放的权重（-0.28，-0.12，*0*。09，0。31）。为说明 SQ2-G4A1S1，选择基础编码 $q = (-2,\ -1,\ 0,\ 1)$、共享附加位 $a_g = 1$ 和移位选择器$r_g = 1$。设固定对齐为 $\ell = 2$，最终缩减为 $\delta = 1$，输出操作数位宽为 $t = 8$。这三种操作得到

$$
\begin{aligned}
z &= 2q + 1 = (-3, -1, 1, 3),\\
v &= (4z) \gg 1 = (-6, -2, 2, 6),\\
P &= \mathrm{clip}_8(v \gg 1) = (-3, -1, 1, 3).
\end{aligned}
$$

这四个值均可用 INT8 表示。使用行输出缩放 0。1 后，它们重构为（-0.3， -0.1，0。1，0。3）。保持基础编码和移位不变，将追加位设为零，得到（-0.4， -0.2，*0*，0。2）。因此，共享位只需占用组内一位，就能使四个重构权重全部移动 0。1。

B 激活量化与数值配置文件

B.1 A4截断与分组校正

对于常规 A4量化，选择逐令牌步长 S ，并构造 a =clip(round(x/S))。在评估实现中使用的动态选择，会在各个截断因子中选出使激活重构误差最小的因子

$$\alpha \in \{0.5,\, 0.625,\, 0.75,\, 0.875,\, 1\}, \qquad S_\alpha = \max(\alpha \|x\|_\infty / 7, \epsilon). \tag{17}$$

编码采用舍入到最近偶数，并使用指定的截断范围。所选缩放因子在令牌间共享。静态变体通过校准获取截断因子，同时保留依赖令牌的幅值估计。我们分别评估静态截断和动态截断。

对于带有分组校正的激活值表示，在普通缩放因子选择后设置 $s = S/16$ 。在此缩放因子下，使用两个共享校正初始化值 0 和 7，拟合 a 和 c 。对于每个初始化值，交替执行经截断的主编码舍入，以及经截断和舍入的组均值残差校正，然后再分别进行一次更新。针对每个组，独立选择平方误差结果较低者。保留最终校正更新之前的主编码。这为公式（12）提供了一种小规模离散搜索；不会针对每个校正候选再次搜索所有截断因子。激活和转换有其单独指定的缩放因子和截断规则。评估所用的激活预处理会在输入通道缩放之前将旋转后的值舍入为 BF16；融合这些阶段会保留这一舍入边界。

B.2 A8和A16执行

A8 使用逐令牌缩放因子对激活值进行量化，并执行采用 INT32 累加的 INT8 乘积，直接使用解码后的SQ操作数 P 。A16 使用指定的 BF16 或 FP16 算术，并按需或通过暂存转换打包权重。对于令牌数量较少的情况，避免激活量化可以抵消更宽算术的成本。此路径无需实体化完整的浮点权重矩阵即可运行。

可选的 A8离群值保护会将选定的大型激活值排除在主要量化乘积之外，并使用原始值加上它们的贡献。均方根阈值、逐令牌上限和校正运算共同决定其准确率与成本。因此，每项结果都会将激活精度和保护机制与权重格式分别标明。

C 跨 GPU 吞吐量测量

	27B SQ4		35B-A3B SQ4	
GPU	预填充	MTP3 解码	预填充	MTP3 解码
RTX A6000	2,723.8	97.0	10,828.8	331.0
RTX 3090	2,915.7	111.5	11,354.8	391.9
RTX 4090	6,098.7	148.9	23,093.3	532.3
RTX 5090	8,221.9	242.6	32,039.0	784.0

表 10：测量 d 四块 GPU 上 SQ4 的吞吐量（令牌/秒）。预填充使用 7,680 个令牌的提示词 .

表 10 报告了图 1 所使用的测量结果。结构化 MTP3 使用 JSONL、CSV 和 SQL 测试固定装置。针对 35B-A3B 检查点的 RTX 5090 测量结果，每个测试固定装置使用五个随机种子。其他解码条目则每个测试固定装置使用一个随机种子。