GC-MoE K3

面向潜在混合专家解码的绿色上下文角色图

Yong Li HaoyuWang Kan Liu TaoLan 阿里巴巴公司
{brucelee.ly, why492961, liukan.lk, tao.lant}@alibaba-inc.com

arXiv:2611.30542v1 [cs.DC] 23 Nov 2026

摘要

Kimi K3 将潜在路由混合专家与全宽共享专家相结合。完整的前馈计算必须对专家进行评分和选择，将令牌投影到潜在空间，在专家并行秩之间交换令牌，计算路由分支和共享分支，并在合并两个分支之前对路由求和结果进行归一化和投影。我们介绍 GC-MoE K3，这是一种面向数据中心 Blackwell 的实现，可将整个计算过程调度为跨两个绿色上下文的 CUDA 图。五个基础角色分别负责输入投影、路由、分发、专家计算和最终处理。分发可以在计算分区评估共享专家的同时，填充路由输入环。最终处理会等待每个令牌所需的专家所有者，并在加载输出投影权重时，重叠执行路由归约和 RMSNorm。条件角色会复用通信分区来执行路由计算，并归约拆分的K 投影部分结果。在 GB200 上，针对 EP8、EP16 和 EP32，在全部 24 种测量配置中，该算子的速度均快于配对的 SGLang 风格参考实现（平均加速比为 $1.1324\times$）。与历史 vLLM 风格测量结果的比较显示，在相同的 24 种形状下，$1.3651\times$ 。两项延迟比较均通过合成 K3 形状输入和残差前缀，对单层图重放进行计时。单独进行的配对数值测试套件通过了全部 142 个案例和 2,936 项秩检查。

1 引言

专家并行（EP）将路由专家的权重分布到不同秩上。对于每个令牌，该层必须确定其专家所有者，将激活值传输给这些所有者，并取回它们的输出以进行加权和。此前的 GC-MoE 工作利用 CUDA 绿色上下文将令牌传输与专家矩阵乘法分离，先后应用于 Hopper 和数据中心 Blackwell，之后又以另一种形式应用于 SM120 [1, 2]。这项工作启发我们将 GPU 资源分别用于通信和计算。K3 改变了这种重叠开始和结束的位置。路由器和共享专家读取完整的隐藏状态，而路由专家读取范围更窄的投影。每个令牌都必须合并其选定专家的响应，然后进行归一化和上投影。

GC-MoE K3 将计时边界从预路由的激活值/路由元组移至整个 K3 前馈子层。输入是注意力后归一化的 BF16 输出，输出则是投影后的路由结果与共享结果之和，可选择是否带有残差前缀。图中的第五种角色负责合并专家响应，并将归一化行流式传入输出投影。另外两种条件角色会将通信 SM 重新用于路由工作，并归约拆分的-K 投影部分结果。我们在 24 种 GB200 配置上评估完整算子，并报告两个基线系列的来源。

2 K3 工作负载与对比路径

2.1 完整的 K3 前馈子层

设 $X \in \mathbb{R}^{T\times 7168}$ 包含单个秩上的归一化的注意力后 FFN 输入。路由器应用一个 $7168 \to 896$ 线性映射。它根据 Sigmoid 分数与特定于专家的校正偏置之和选出排名前 16 的专家，然后在进行任何路由缩放之前，对选出的 Sigmoid 分数进行归一化。另一个独立的 $7168 \to 3584$ 投影会为所有选出的路由专家生成潜在激活 Z。每个路由专家通过两个输入投影、一个 SiTU 门控和一个输出投影，对 Z 的一行进行变换，构成一个 $3584 \to 3072 \to 3584$ 多层感知器。对于门控值 g 和上投影值 u，所评估的 SiTU 为

$$\phi(g,u) = \big[4\tanh(g/4)\,\sigma(g)\big]\big[25\tanh(u/25)\big]. \tag{1}$$

两个共享专家分别在 X上计算一个 $7168 \to 3072 \to 7168$ SiTU 多层感知器。实现将它们的中间通道拼接成一个宽度为 6144 的共享多层感知器，因此其输出投影会将两者的贡献相加。将它们在令牌 t 处的和记为 S_t，将该令牌的 Top16 集合记为 $\mathcal{E}_t$，将其路由权重记为 $a_{t,e}$，并将路由专家记为 f_e。令 c 为路由缩放因子（基准测试默认值为 1）。在实数运算中，输出为

$$R_t = \sum_{e\in\mathcal{E}_t} a_{t,e} f_e(Z_t), \quad U_t = \mathrm{RMSNorm}(cR_t)W_{\mathrm{up}}^{\top}, \quad Y_t = U_t + S_t + P_t. \tag{2}$$

P_t 是可选的全宽残差前缀；不存在时将其省略。RMSNorm 会对合并后的行在其 3584 个通道上进行归一化，然后应用学习到的逐通道权重。测试框架使用 $\epsilon = 10^{-6}$。W_{up} 由所有路由专家共享。在 896 个路由专家的情况下，EP8、EP16 和 EP32 分别为每个秩分配 112、56 和 28 个完整的路由专家。共享专家在我们的基准测试中为本地专家，并且会被复制。

2.2 SGLang 和 vLLM

两种框架路径都调用 DeepGEMM Mega MoE 来执行潜变量路由计算，并单独计算全宽共享分支 [3, 4, 5, 6]。图1和2展示了各基准测试封装器在何处分流并重新汇合其辅助流。

SGLang。SGLang 会根据设备和形状，在合并式、双流式和独立前置投影之间进行选择。在所有测量形状下，封装器都会使用一个 BF16 权重 GEMM 来计算路由器 logits 和潜在下投影。Triton 尾处理阶段会选择 Top16，并将潜变量输出转换为 BF16。第二个 Triton 步骤会为 `sgl-deep-gemm` 0.2.8打包 FP8 激活、K32 缩放因子和路由信息。辅助共享多层感知机会根据形状使用 FlashInfer、TGV 或 PyTorch 投影。默认情况下，封装器将 Mega MoE 可用的流式多处理器（SM）数量限制为比设备总数少两个。

vLLM。 测得的封装采用图2中的标准 BF16 调度方案。路由专家使用固定版本的 vLLM DeepGEMM 分支，Top16、暂存、共享 SiTU 和路由 RMSNorm 则使用本地 Triton。小批量 BF16 门控为可选项。已存档的测量结果未标明各个形状所用的门控。对于符合条件的捕获 TP1+EP 批次，另一种 K3.1 服务调度方案会将共享 W1 与潜在下投影打包，并在旁路流上运行共享 W2。

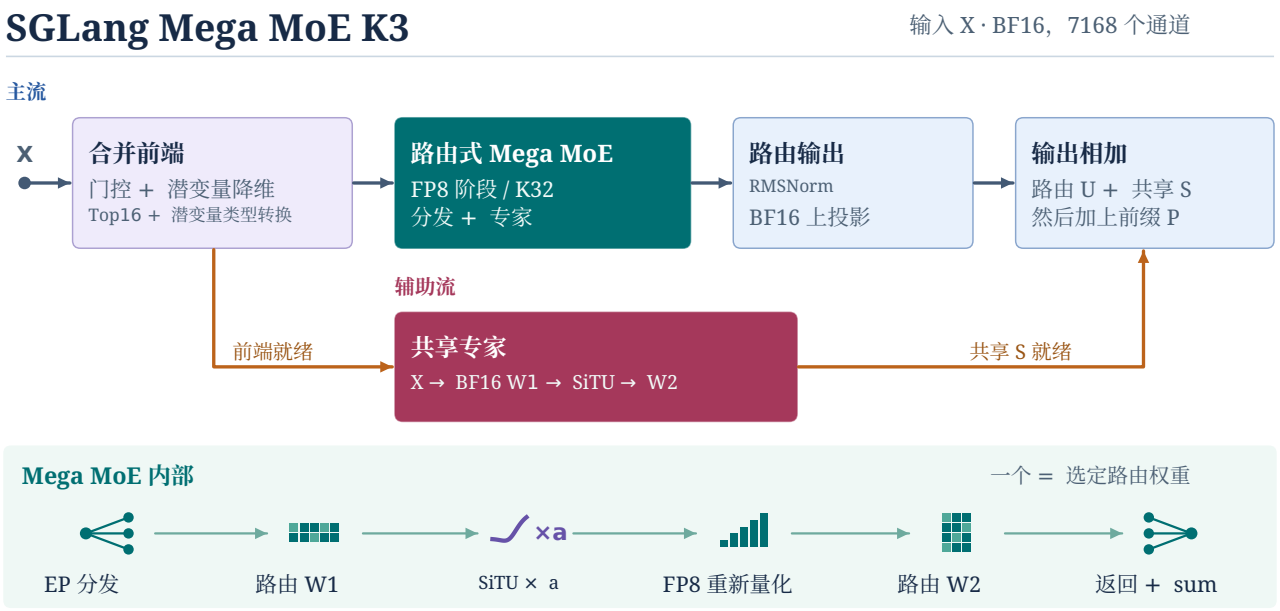

图 1：针对测得的解码形状的 SGLang 风格调度。辅助流在合并前端之后启动共享多层感知机，并读取原始输入 X。主流暂存潜在输入，运行路由式 Mega MoE，然后对路由求和进行归一化和投影。最终加法操作需等待共享结果 S。琥珀色箭头表示跨流顺序。

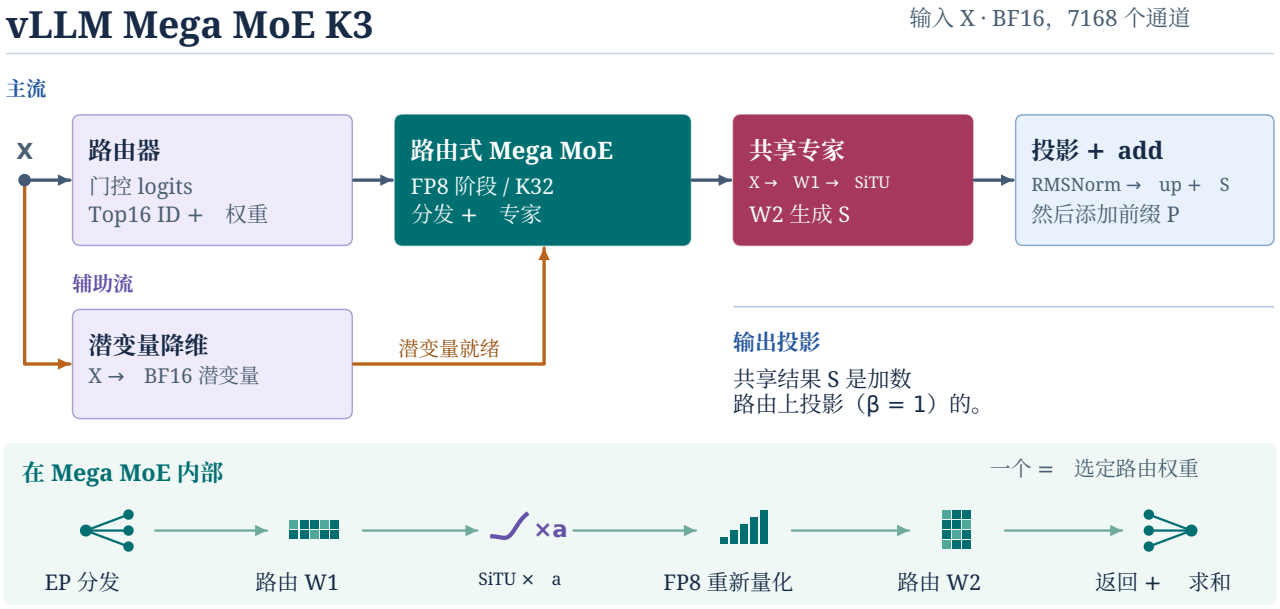

图 2：针对相同形状的 vLLM风格调度。主流上的门控和 Top16 操作可以与辅助流上的潜在下投影重叠。各流在暂存之前汇合。随后，Mega MoE 和共享多层感知机依次在主流上运行。BF16 上投影累加到 S中，另有一个独立的加法操作用于应用残差前缀 P。

表1：在相同的注意力后输入和包含前缀的输出边界下比较的调度方案。这两行 Mega MoE 描述的是基准测试封装器。

Path	前端	路由专家和共享专家	输出
SGLang 风格	合并门控与下投影，然后 Top16 和 FP8 暂存	Mega MoE 路由调用，共享旁路流上的多层感知器	路由归一化和上投影投影，然后合并并添加共享/前缀
vLLM风格	Gate/Top16 可以重叠潜在下投影，然后暂存	Mega MoE 路由调用，然后在主流上执行共享多层感知机	路由归一化，BF16与共享加项进行投影，然后进行前缀加法
GC-MoE K3	联合前端投影，使用潜在量化，然后路由器	流分发和四阶段专家计算调度器在独立的 SM 分区上	对等就绪合并、归一化、上投影和共享/前缀加法

3 角色图

我们介绍提交 `53f430914ea8f3de196d9a53ba8b9750ca6ba8d6` [7]中的实现。对于正的物理行数 T，基本角色为前端 (1)、路由器 (2)、分发 (3)、计算 (4) 和 Finalize (5)。运行时会将计算 SM 请求向下取整至设备的协同调度对齐粒度，并通过 CUDA 驱动程序获取分区。其余 SM 构成通信资源。在 EP 数量不超过 8 时，默认分发 请求 为 8 个 SM；EP 数量更大时则为 16 个 SM。对齐后的划分决定了实际的 SM 数量。在支持的情况下，实现会使用均衡的工作队列资源 [8]。两个分区能够并发推进。实际的物理并发还取决于设备的工作队列、驻留性，以及两个分区发出的指令。

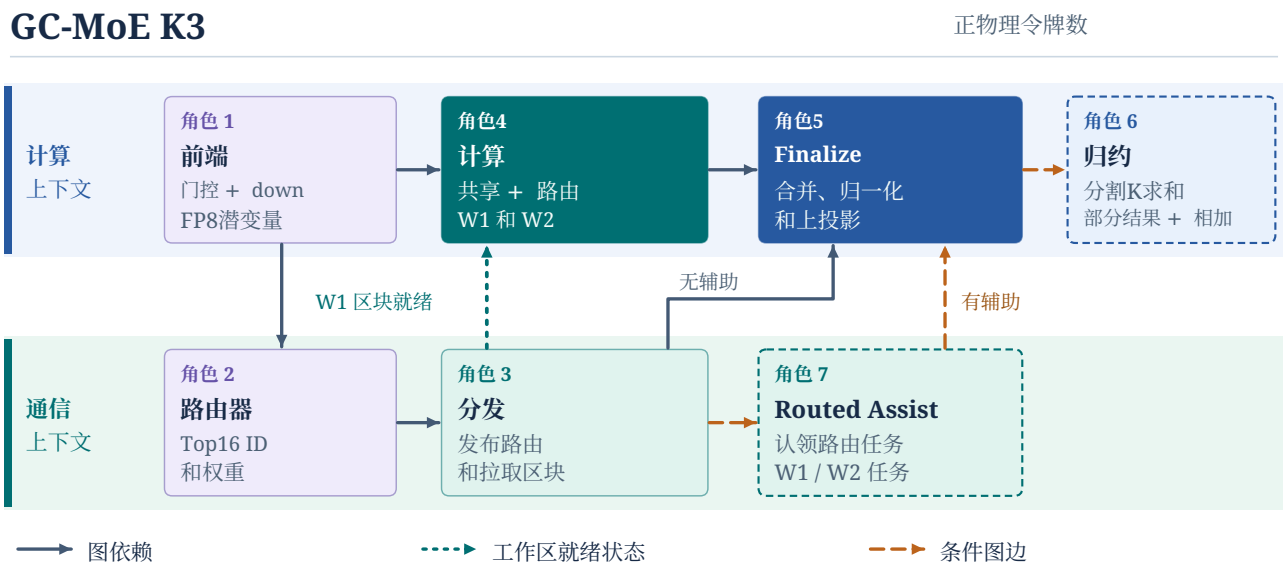

图 3：GC-MoE K3 在两个绿色上下文中的依赖关系。前端通常会在计算开始前完成。早期共享计划可以更早触发计算。虚线边将输入块就绪状态传递给路由 W1。共享 W1 可以独立进行。无辅助时，Finalize 汇合分发和计算。有辅助时，经过角色 7 的路径取代分发到 Finalize 的直接边。Split-K 投影可启用角色 6。

在 Green Context 路径中，前端、计算、Finalize 和可选的投影 Reduce 使用计算上下文。路由器、分发和可选的 Routed Assist 使用通信上下文。图 3 展示了依赖图。分发始终在路由器之后执行。计算通常在前端之后执行，也可以独立于路由器开始共享工作。选择早期共享计划时，Front 到 Compute 的边采用程序化方式，前端会提前触发依赖启动。计算中经路由的部分仍受已发布元数据和环形数据的门控。Finalize 会汇合本地分发与计算的完成状态。若加入了 Routed Assist 节点，Finalize 则改为汇合 Assist 与计算，因为 Assist 已在分发之后执行。当上投影使用两个分区时，投影 Reduce 是 Finalize 的后继节点。 K 主上下文诊断模式使用从路由器到计算的完成边，并且不提供 Green 早期共享或 Assist 变体。

零物理行秩会执行分发、计算和 Finalize，以服务远程令牌并释放其工作区。它没有本地前端、路由器或投影任务。对于固定的正物理图形形状和较小的逻辑活动计数，路由器会将前缀外路由标记为无效。该算子为活动行定义输出。这些情况区分了空源秩与整个 EP 组都没有任何任务的情形。

4 角色实现与重叠

4.1 角色 1 和 2：组合前端投影与路由发布 n

前端读取 X以及一个形状为 $[896+3584, 7168]$的 BF16 矩阵，其中包含路由器权重和下投影权重。两个线程块（CTA）组成一个集群。它们使用张量内存加速器（TMA）暂存操作数，执行 BF16 张量核心运算，并在张量内存（TMEM）中累加。尾处理阶段将门控列存储为 FP32 对数几率。对于潜在列，它会将值舍入为 BF16，然后立即将 E4M3 激活值以及每 32 个元素对应的一个 UE8M0 指数写入对称工作区。缩放因子为组绝对最大值（下限为 10^{-10}）除以 448 后，再向上舍入到 2 的幂。这样无需在暂存前单独发布稠密潜在张量。原始输入仍可供独立的共享分支使用。

路由器为每个物理行分配一个 256 线程 CTA。线程扫描对数几率面板，计算带校正偏置的 Sigmoid 分数，并选出线程束候选项。最终线程束合并这些候选项。它会优先选择较小的 ID 来解决平局，按递增的专家 ID 顺序写入获胜项，并在启用路由重归一化时（如基准测试中所示）对其无偏 Sigmoid 分数进行归一化。可选的有效令牌标量会控制此选择。掩码行会写入无效 ID 和零权重。分发仅读取有效路由。

规划器根据启动几何配置确定早期共享的适用条件。它为前端预留集群容量，并检查剩余容量是否至少能容纳共享 W1 行块一半的任务（上限为计算集群数）。当切片数超过流水线深度的两倍时，前端在发出两轮 TMA 加载后即可触发计算。共享 W1 可以与 X 和本地共享权重一同启动。计算仍需等待 Front 依赖后才能返回。该几何配置允许部分共享波次与前端重叠。

4.2 角色 3：路由元数据和分块式负载交换

每个源秩统计本地路由，并将令牌槽索引写入互不重叠的目标区间。单 CTA 路径在一次遍历中完成计数和槽位分配。EP 组将一个连续的专家 ID 范围分片，因此专家 ID 可同时标识所有者秩和本地专家索引。对于不超过 32 个秩的组（包括所有已测配置），源会通过带纪元标记的就绪信号发布其计数和索引。接收方获取各源的数据，并在本地发布已完成的每位专家的总数。符合条件的小型形状使用单个 CTA 执行源路由遍历，从而避免

多 CTA 遍历所需的网格协调。对于更大的组，该实现还提供按专家进行的系统原子发布路径。

目标端使用计数将行的逻辑块分配给各个专家。在专家边界处对块进行填充，并将其映射到有界的激活环形缓冲区。加载器线程束分阶段通过共享内存提取令牌字节，并复制缩放字和源秩/令牌/槽位元数据。存储载荷和元数据后，分发以释放语义递增对应路由 W1 块的就绪计数。部分块中的最后一个令牌会将未使用的行数计入该计数。分发仍在填充其他块时，计算工作线程即可开始处理已完整的块。环形缓冲区回绕时，代际计数器可防止覆盖。分发会等待 W1 释放输入槽，而 W1 会等待 W2 释放中间槽。

4.3 角色4：在同一任务调度中执行共享计算与路由计算

专家任务会标识一个 M块、N分块、层阶段、本地专家和有效行数。主 CTA 的调度器会将 32 字节描述符写入每个 CTA 共享内存中的同一交替槽位。第一个线程束组包含调度器、TMA 加载和 MMA工作线程。另外两个线程束组负责执行尾处理阶段。事务屏障负责管理共享内存操作数阶段和 TMEM累加器交接。路由计算使用操作数互换的 SM100 分块缩放`tcgen05` MMA，处理 FP8 激活和 FP4权重。共享任务则运行 BF16 MMA，不使用微缩放操作数。离线变换会交错排列 W1门控/上投影行，并按 N和K切片打包 W1 和 W2 载荷。FP4 载荷切片跨越 128 个 K 元素，BF16 载荷切片跨越 64 个。打包后的权重缩放值则遵循张量核缩放布局。混合格式指令将 FP4 权重作为操作数 A，将 FP8 激活作为操作数B。在执行分块缩放指令之前，缩放分块会移入 TMEM。

路由输入块就绪之前，调度器就可以发出共享 W1。这里，W1计算门控和上投影，W2则将门控后的激活值映射回分支输出宽度。根据形状不同，它可以在共享 W1之后立即发出共享 W2，也可以在初始路由 W1任务之后或路由遍历之后发出。对于符合条件的小型共享形状，它可以在共享 W1生成输入的同时，将共享 W2加入队列。W2加载阶段会等待与之匹配的 W1 K64切片，而其他形状则采用粒度更粗的就绪计数。路由 W1会将门控和上投影结果舍入为 BF16，计算 SiTU，并写入 K32 FP8中间数据。W2可以开始读取中间块中已就绪的切片。每次 K128加载都要等待两个相关的 W1完成位，预期掩码则会根据环形缓冲区代际进行调整。当实时路由集合能够容纳在环中时，调度器可以批量执行 W1遍历。若环会回绕，则调度器会交错执行 W1和W2工作，以便后者释放槽位。路由 W2会通过对齐向量存储，将 BF16值散射写入源令牌对应的秩。每个返回槽包含一行3584个元素，其地址根据 top-k槽、源行和通道偏移计算得出。共享 W2会在本地写入全宽BF16结果。

角色 7：条件路由辅助。 当通信分区适合双 CTA 组且路由几何结构提供额外工作时，规划器会考虑辅助。运行时会检查实际的通信上下文启动配置下是否至少有一个集群能够驻留。通过准入检查的辅助会在分发之后启动，并从与计算相同的队列中获取路由的 W1/W2 任务。它不执行任何共享多层感知机任务；如果实时路由集合会绕回环，则不会获取工作。在完成主要任务后，这种方式可以重新利用通信 SM，同时避免改变环进度条件。如果准入检查拒绝辅助，五角色图仍然有效。

4.4 角色 5 和 6：对等方就绪终结与上投影

本地工作线程完成后，第一对获取发布权的线程块中会选出一个 CTA，将该秩的返回纪元广播给所有对等方。接收方仅会针对令牌的有效专家槽位所属方检查该纪元。所有选择同一所属方的令牌都会检查同一项发布内容。Combine 线程束按槽位加载 BF16 响应，与存储的路由权重相乘，并按照固定的槽位顺序以 FP32 累加。随后将潜变量和舍入为 BF16。非单位路由缩放因子会乘以该潜变量和，并在 RMSNorm 之前引入第二次 BF16 舍入。归一化后的潜在向量行以 BF16 写入，并通过行就绪信号释放。无效槽位不产生任何贡献。

两个 CTA BF16 投影工作线程预填充权重阶段，同时合并线程束生成行。投影会在匹配的行就绪后发起潜在向量 TMA 加载，并计算 $3584 \to 7168$这些归一化行的扩展。双槽累加器流水线使下一个投影分块能够在尾处理阶段完成存储时使用 TMEM。尾处理阶段会加上本地共享输出。当选定的分块几何形状和可用的部分结果缓冲区有利于拆分-K时，Finalize 会写入两个 FP32 投影部分结果。角色 6 对它们求和，并执行相同的共享加法和可选的前缀加法。它会增加一个输出依赖，该依赖由图 3 中的最终完成边表示。

对于包含前缀的输出，GC 会对 FP32 投影进行舍入，并按照 bf16(bf16(bf16(U) $+ S$) $+ P$) 的顺序执行两次 BF16 加法。不含前缀时，它会在舍入前将 S加到 FP32 投影上。这些边界对于与 vLLM风格的 beta 加法和独立的前缀加法进行比较十分重要。在测得的 Mega MoE 配置中，W1 尾处理阶段会先将路由权重应用于 SiTU 输出，然后再将其量化以供 W2 使用。在 GC 中，它会在 Finalize 归约中将返回的专家贡献相乘。线性 W2 在实数运算中允许这两种位置。量化和舍入会使有限精度计算产生差异。评估会将所有路径与相应的数值参考进行比较。

4.5 工作区重放与排序

图完成和数据可见性由不同机制处理。本地图边对角色启动和汇合进行排序。释放/获取握手会发布分发元数据、环就绪状态、返回的对等方纪元和归一化行。工作区使用两个接收计数代次。Finalize 会推进持久奇偶标记，并清除已完成调用所使用的本地任务和环状态。随后执行的分发会在发布新代次之前清除已退役的接收计数存储区。计数协议的完成（包括零计数专家）会对未来的重用进行排序。对等方返回纪元保护逐令牌贡献读取，并区分过期发布。该协议在保留分发中的通信和 Finalize 中的对等方等待的同时，移除了尾部的跨秩工作区屏障。

运行时会缓存已编译角色二进制文件、TensorMaps 和生成的子图。在捕获期间，它会将子图附加到调用方的图。即时启动会实例化缓存的子图。捕获的图对象会保留其内核所使用的存储。共享同一工作区分配的调用和重放次数必须串行化。这些机制适用于所提供的图与形状配置。我们使用全算子测量进行总体延迟比较。

5 评估

5.1 设置与来源

基准测试报告包含在 NVIDIA GB200（报告的设备名称为 L20A）上针对 EP8、EP16 和 EP32 [9]的测量结果。对于每种 EP 大小，我们使用每个秩的对称令牌数量 1、2、4、8、16、32、64 和 128。测试框架使用固定随机种子和 Kimi K3 的形状生成权重、隐藏状态以及每个令牌的残差加数。路由权重使用 FP4，路由激活值使用 FP8，并按每组 32 个元素使用

缩放因子。共享多层感知机和全宽投影使用 BF16 权重。测量区间涵盖从准备就绪的注意力后隐藏状态到包含前缀在内的最终输出。注意力、请求处理、权重转换、工作区分配以及首次图捕获均不在计时区间内。

基准测试捕获一次算子调用。在每个秩上，将每个窗口的 CUDA 事件时间除以该窗口的重放次数，然后取各次重放估算值的中位数。报告的延迟为各秩中位数中的最大值。基准测试默认使用 10 个预热窗口、30 个测量窗口，以及每个窗口 20 次重放。每个计时窗口开始前的屏障、缓存驱逐和同步均不计入事件时间间隔。源代码会在不同令牌数量之间交替使用两种计时顺序。

配对及历史数据。 GC 和 SGLang 的 24 个数值，是基准测试报告中基于 PyTorch 2.13 的同轮测量结果。报告中的 vLLM 数值则是沿用自此前 PyTorch 2.14 运行的历史测量结果。两个比率均以当前 GC 延迟为分母。因此，与 vLLM 的比较涉及两个不同的软件环境。

5.2 测量延迟

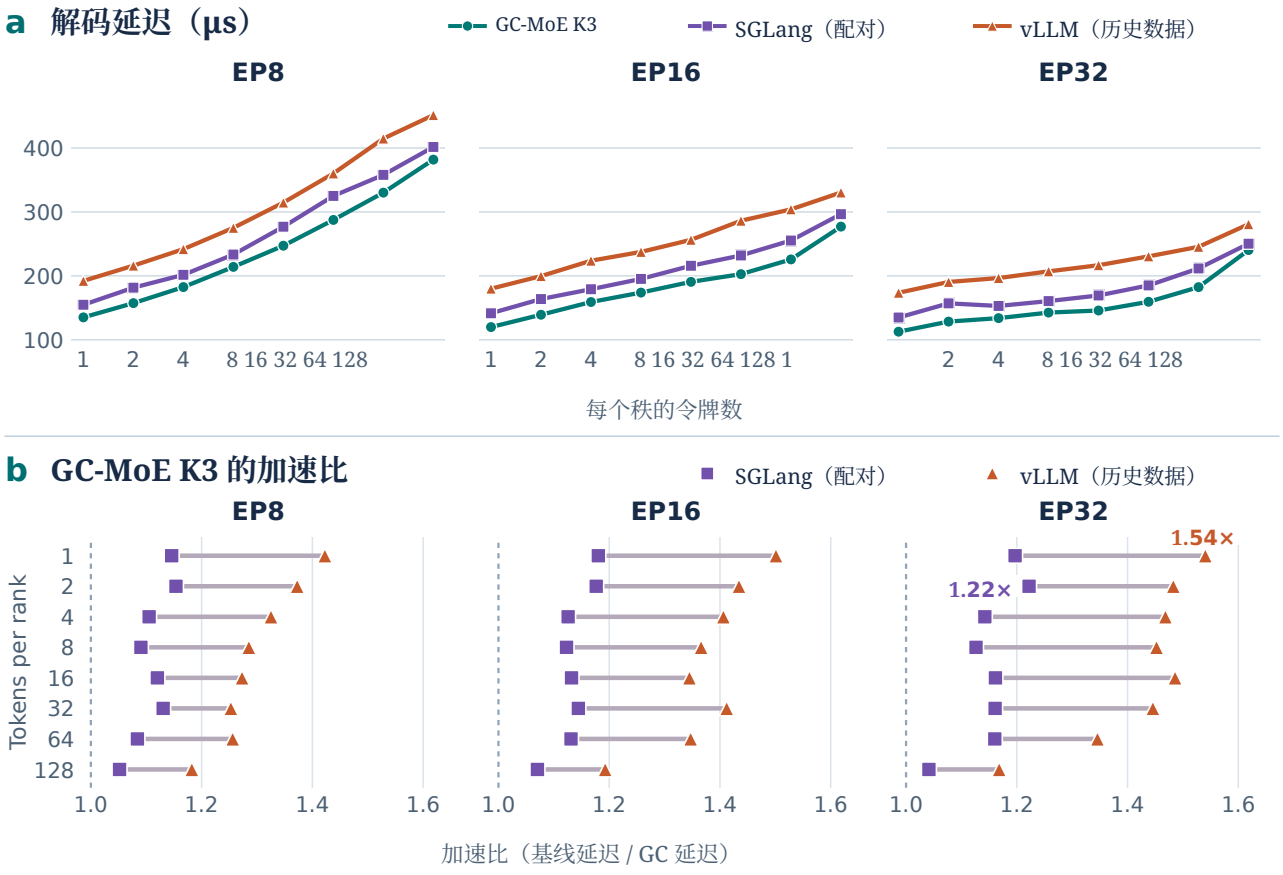

图 4：GB200 在 24 种 EP/令牌配置下的解码。**a,** 曲线连接连续令牌数量下测得的各秩中位数延迟最大值。**b,** 标记显示相同 EP 大小和令牌数量下的基线/GC 延迟比。虚线表示相等线（1×），标签标出各基线的峰值比。GC 和 SGLang 共用一次测量运行。vLLM 使用历史测量数据。

表2：根据实测延迟计算的加速比。每个平均值均为各点基线/GC比值的非加权平均值。与两个对比序列相比，GC 在列出的每个点上都更快。

		SGLang / GC（配对）		vLLM / GC（历史）	
组	点数	Mean	Peak	Mean	Peak
EP8	8	1.1102	1.1537	1.2962	1.4226
EP16	8	1.1354	1.1804	1.3756	1.5011
EP32	8	1.1514	1.2222	1.4236	1.5404
All	24	1.1324	1.2222	1.3651	1.5404

配对的 SGLang 加速比在 $1.1102\times$、$1.1354\times$ 和 $1.1514\times$，分别对应 EP8、EP16 和 EP32（表 2）。配对加速比峰值为 $1.2222\times$，出现在 EP32 且每个秩有两个令牌时（128.456 对比 156.994 μ 秒）。在 EP32 且每个秩有 128 个令牌时，差距缩小至 $1.0413\times$（240.379 对比 250.310 μ秒）。历史 vLLM 比率的峰值为 $1.5404\times$，出现在 EP32 且每个秩有一个令牌时。所有点数的相应平均值为 $1.1324\times$ 和 $1.3651\times$。在每个秩的令牌数固定时改变 EP，会同时改变本地专家数量和全局令牌数。

5.3 数值精度

每个比较套件都包含默认、边界和非对称秩输入，包括零行秩。EP8、EP16 和 EP32 套件分别涵盖 37、45 和 60 个案例。检查器会在有无前缀的情况下测试图重放和即时调用。参考实现会对量化操作数进行解码，并模拟各路径的中间舍入和输出加法顺序。在 vLLM风格的 RMSNorm 中，归一化后的值会先舍入为 BF16，再应用归一化权重。GC 则先应用归一化权重，再对归一化结果进行舍入。验收阈值为：与相应参考实现的余弦相似度至少为 0.99，相对误差至多为 0.02 ℓ_2；算子之间直接比较时也采用相同的余弦相似度阈值。检查器还会验证掩码行的专家ID无效且路由权重为零。
GC/SGLang 配对实验通过了全部 142 个案例和 2,936 项秩检查。历史上的 GC/vLLM 套件也通过了同样数量的测试。表 3 汇总了各自套件中报告的最差参考比较结果。

表3：与各路径参考值对比的数值结果。GC 和 SGLang 的数据来自配对实验。vLLM 的数据来自历史记录。

Path	最小余弦值	最大相对 ℓ_2	最大绝对误差
GC-MoE K3	0.999995410	0.003025480	0.50
SGLang 风格	0.999996305	0.002748258	0.50
vLLM风格	0.999996006	0.002818683	0.50

局限性。 合成输入和权重可能会产生与部署的检查点不同的专家偏斜和缓存行为。延迟结果涵盖算子边界处的对称输入。非对称输入用于数值验证。要评估各个角色的影响和服务性能，还需要进行匹配消融实验和系统级测量。

6 相关工作

GC-MoE 针对常规专家并行 MoE 引入了绿色上下文分离，将令牌分发与分组 GEMM 分开 [1]。其 SM120 扩展将该流水线适配到线程束级矩阵指令和 PCIe [2]。本文面向 SM100 系列以及 K3 更广泛的算子边界；在此边界下，前置投影、路由归一化和投影都与专家 MLP 存在重要依赖关系。SGLang 和 vLLM 路径采用 Mega MoE 方法，在一次设备调用中合并路由交换与专家 MLP [3, 4, 5, 6]。这两种框架路径都在该调用之外调度共享多层感知机和输出投影。GC 则将它们与路由计算安排在同一角色图中。

7 结论

GC-MoE K3 从注意力后输入开始，对 Kimi K3 子层依次执行路由、专家计算和归一化投影和。Dispatch 将就绪的输入块发布给 Compute。专家结果写入后，每个所有者秩发布一个返回纪元，供 Finalize 检查。调度器可以将共享 W1 与前端及通信工作重叠，并在令牌交换后复用分发分区。Finalize 合并路由贡献，并使用令牌特定对等等待，为 BF16 上投影准备各行。在 GB200 扫描中，GC-MoE K3 在全部 24 个测量点上均优于配对的 SGLang 风格路径。在包含前缀的评估边界处，其平均配对加速比为 $1.1324\times$ 。

参考文献

[1] Yong Li、Kan Liu 和 Tao Lan。《GC-MoE：通过 CUDA 绿色上下文，使专家并行令牌分发与 FP8/FP4 分组 GEMM 重叠》。arXiv:2611.04222，2026年。 https://arxiv.org/abs/2611.04222。[2] Yong Li、Haoyu Wang、Kan Liu 和 Tao Lan。《SM120 上的 GC-MoE：在消费级 Blackwell 上，以 warp 级 FP8xFP4 分块缩放 MMA，使专家并行令牌分发重叠》。arXiv:2611.16219，2026年。 https://arxiv.org/abs/2611.16219。[3] SGLang 贡献者。SGLang 中的 Kimi K3 模型实现，2026年。修订版 da395ae091。模型源码。[4] SGLang 贡献者。SGLang DeepGEMM：Mega MoE 实现，2026年。版本 0.2.8。固定源码。[5] vLLM 贡献者。vLLM 中的 Kimi K3 NVIDIA 模型实现，2026年。修订版 89d9b4d187。模型源码。[6] vLLM 贡献者。vLLM 使用的 DeepGEMM 分支：SM100 SiTU Mega MoE，2026年。修订版 44cba7d。固定源码。[7] GC-MoE 贡献者。GC-MoE K3 源码和评测框架，2026年。实现修订版 53f4309。项目仓库。[8] NVIDIA 公司。CUDA 驱动程序 API：绿色上下文，2026年。 驱动程序 API 参考。[9] GC-MoE 贡献者。GB200 Kimi K3 在 EP8、EP16 和 EP32 下的解码延迟与数值精度，2026年。修订版 53f4309 的测试报告。源码包包含原始报告，位于 `anc/data/`。参见附录 A 查看实测延迟。

A 实测延迟值

表4 列出了图4中使用的实测延迟。源码包包含原始基准测试报告，以及用于根据完整精度条目重新生成图4 和延迟表的脚本。表中数值以微秒为单位，保留三位小数。

表 4：秩中位数延迟的最大值（μ秒）。GC 和 SGLang 是 PyTorch 2.13 下同一轮次的数值。vLLM 的数值是 PyTorch 2.14 下的历史数据。

EP	令牌数/秩	令牌总数	GC-MoE K3	SGLang（配对）	vLLM（历史版本）
8	1	8	135.065	154.775	192.141
8	2	16	157.357	181.541	215.983
8	4	32	182.449	201.645	241.794
8	8	64	214.038	233.386	275.104
8	16	128	247.180	276.877	314.660
8	32	256	287.506	325.036	360.202
8	64	512	330.219	357.984	414.777
8	128	1024	381.850	401.629	451.474
16	1	16	119.879	141.504	179.954
16	2	32	139.171	163.740	199.621
16	4	64	159.087	179.103	223.705
16	8	128	173.848	195.220	237.470
16	16	256	190.626	215.784	256.377
16	32	512	202.691	231.941	286.224
16	64	1024	225.721	255.298	304.059
16	128	2048	277.039	296.546	330.424
32	1	32	112.664	134.846	173.552
32	2	64	128.456	156.994	190.450
32	4	128	133.874	152.922	196.571
32	8	256	142.553	160.566	207.027
32	16	512	145.797	169.329	216.627
32	32	1024	159.448	185.091	230.522
32	64	2048	182.385	211.616	245.426
32	128	4096	240.379	250.310	280.806

B 复现工作负载

以下命令需要第 3 节中确定的实现修订版本。该脚本`scripts/run_k3.sh`会选择 Kimi K3 形状和默认的八组令牌数量。在`bash scripts/run_k3.sh 8 both --baseline sglang` SGLang 依赖环境中运行。替换为 16 或 32，并设置脚本的节点/秩配置，以指定所需的 EP 组。SGLang 封装器需要 `sgl-deep-gemm` 0.2.8，并使用 FlashInfer 执行 GEMM 和 RMSNorm。对于 vLLM，请使用 `--baseline vllm`，以及第 2.2 节所引用的 DeepGEMM 修订版本。请使用单独的依赖环境，因为两个构建都会导出 `deep_gemm`，且必须与活动的 PyTorch 应用二进制接口匹配。对于任何新的测量，请记录门控实现、软件包版本、CUDA 驱动程序/运行时、节点拓扑、物理 SM 分配和图形形状。需要新的匹配环境测量，以消除历史比较注意事项。