arXiv:2601.05296v1 [cs.LG] 8 Jan 2026

MoEBlaze：打破内存墙，在现代 GPU 上实现高效 MoE 训练

张济远 [1] **刘一宁** [1] **严思齐** [1] **邓立森** [1] **曹珍妮** [1] **杨淑琪** [1] **倪敏** [1] **薛碧** [2] **李深** [1]*Meta Platforms Inc*，[2]*Thinking Machines Lab*

摘要

普遍存在的“内存墙”瓶颈在现代大规模混合专家（MoE）架构中显著加剧。MoE 固有的架构稀疏性不仅导致稀疏算术计算，还会带来显著的激活内存开销——这主要源于大型令牌路由缓冲区，以及物化和缓冲中间张量的需求。这种内存压力限制了 GPU 上可容纳的最大批量大小和序列长度，也导致过多的数据移动，进而妨碍性能提升和模型高效扩展。我们提出 MoEBlaze，这是一种内存高效的 MoE 训练框架，通过协同设计的系统方法解决这些问题：(i) 采用优化数据结构的端到端令牌分发与 MoE 训练方法，以消除中间缓冲区和激活物化；(ii) 协同设计内核并结合智能激活检查点，以减少内存占用，同时实现更优性能。我们证明，与现有 MoE 框架相比，MoEBlaze 可实现超过 $4\times$ 的加速比，并节省超过 50% 的内存。

1 引言

过去几十年来，处理器吞吐量的提升速度远超内存带宽和延迟的改善，形成了长期存在的“内存墙”，不断扩大计算与数据移动之间的差距（Wulf & McKee, 1995）。实际上，这种差距意味着，即使拥有充足的算术单元，端到端吞吐量也往往受限于参数和激活值的读取、写入与交换速度（Williamset al., 2009）。

混合专家（MoE）架构通过稀疏激活，使万亿参数模型能够以可控的训练成本运行，从而重塑了大规模深度学习（Shazeer 等人，2017；Kaplan 等人，2020年；Hoffmann 等人，2022年）。然而，带来这些增益的稀疏性也同时降低了计算密度，因为每个词元只有一部分专家处于激活状态。当这种架构稀疏性与大语言模型（LLM）分布式训练的规模相结合时，现代系统中的内存压力会显著加剧。当模型超出单个设备的高带宽内存（HBM）容量时，训练必须分布到更多 GPU 和节点上，从而加大对设备内存带宽和互连吞吐量的压力。随着序列长度和批次大小持续增长，性能很快便不再受原始浮点运算次数的限制，而是受系统内存和通信子系统的制约。因此，直接降低内存占用并提升端到端的有效带宽利用率，对于打破 MoE 训练中的内存墙并实现

高效的模型扩展

尽管参数存储往往备受关注，但在训练过程中，激活内存同样是造成内存墙的重要因素。在最先进的大语言模型训练达到万亿词元规模时（Brown 等人，2020年；Touvron 等人，2023年；Team 等人，2024），更长的序列、更大的批次以及更复杂的路由机制，会导致整理、重排和暂存中间张量所需的内存缓冲区大幅增加。因此，这些激活缓冲区会占用 GPU 内存占用和带宽的很大一部分，直接限制系统能够处理的最大批量大小和序列长度，进而制约模型的可扩展性和训练效率。

为解决这一系统瓶颈，早期方法依赖令牌丢弃或填充等启发式策略，以限制和管理激活缓冲区（Samuel, 1959；Fedus et al., 2022），但这往往以牺牲模型稳定性为代价。较新的系统则着重优化稀疏专家计算的计算与通信复杂度（Gale etal., 2023；Aminabadi etal., 2025）。然而，令牌分发所需的辅助激活缓冲区，以及填充或物化中间结果的要求，仍占整体模型内存占用的很大一部分。

为解决这些局限，我们提出了 MoEBlaze，这是一种内存高效的混合专家训练框架，能够在不牺牲准确率的前提下，大幅提升混合专家训练的内存效率，同时实现更高的训练

吞吐量。具体而言，我们针对激活内存瓶颈的两个主要来源：(i) 令牌路由，传统实现会为每个专家分配大型辅助缓冲区，以整理并存储激活值；(ii) SiLU 和 SwiGLU 等现代非线性函数所加剧的中间激活存储（Shazeer， 2020年；Ramachandran 等人，2017；Elfwing 等人，2017）。MoEBlaze 旨在有效突破内存墙，充分发挥现代 GPU 架构的效用，从而提升吞吐量。我们的贡献如下：

- 我们提出了一种高效的端到端令牌分发与训练方法，可显著减少用于令牌路由和激活物化的中间激活缓冲区。我们的方法无需填充或令牌丢弃，在不牺牲准确率的情况下减少内存使用和数据移动，同时实现更高的计算效率。

- 我们为上述内存高效的计算方案设计了高效的数据结构和算法，能够充分利用 GPU 的大规模并行性和高带宽，并避免复杂的多内核流水线。

- 我们将训练内核与智能激活检查点方案协同设计，在 GPU 上实现更高计算效率的同时，进一步缓解现代复杂激活函数所带来的显著内存占用。

- 总体而言，与其他先进的 MoE 训练框架相比，我们的方法在各种混合专家基准测试中可实现超过 $4\times$ 的加速比，并节省超过 50% 的内存。

2 背景与动机

本节回顾 MoE 训练的背景，并指出当前制约其性能的关键系统瓶颈。

我们首先定义全文中使用的符号。本文主要关注令牌选择型混合专家，因为它已在生产环境中得到广泛应用。MoE 的计算从令牌输入开始，该输入表示为向量 $\mathbf{x} \in \mathbb{R}^{L\times d}$，其中 L 表示一个步骤中路由的令牌实例数（例如，批量大小 $\times$ 序列长度）， K 表示每个令牌选中的专家数， E 表示专家数， d 表示模型维度。

2.1 门控网络与令牌路由

门控网络决定将每个输入令牌路由到最相关的专家。该网络通常是一个线性变换，将输入维度 d 映射到专家数量 E，从而为每个令牌生成一个专家分数。随后进行 Top-K 选择，为每个令牌收集门控分数最高的专家所对应的专家 ID。输入 $\mathbf{x}$的门控输出定义如下：

$$\text{topk_experts} = \text{TopK}(\text{softmax}(W_g\mathbf{x}))$$

其中， $W_g \in \mathbb{R}^{E\times d}$ 是门控网络参数，K 是每个令牌选中的专家数。结果*topk experts*是每个令牌选中的专家 ID 列表。

完成 Top-k 选择后，必须将令牌实际路由到对应专家的执行缓冲区。在传统实现中，这一过程需要大量辅助内存和额外处理，以压缩并存储分发的令牌，从而形成一个关键的内存瓶颈。

早期研究（如 Switch Transformers 和 GShard（Fedus 等人，2022年；Lepikhin 等人，2021年）采用容量受限路由（令牌丢弃）机制来管理令牌分发。令牌按其门控分数排序，并打包到专家 e的缓冲区中；任何超出 C 的令牌都会被丢弃或路由至残差路径。容量的一种典型取值为：

$$C \approx \gamma \cdot \frac{Bk}{E}.$$

其中， γ是用户定义的容量因子。由于使用固定大小的缓冲区，容量受限路由易于在系统中实现，但代价是模型质量下降。

近期文献关注无丢弃路由机制（Rajbhandari 等人，2022年；He 等人，2021年），这类机制通常能带来更好的模型质量。该方法确保每个令牌都由某个专家处理，从而提升模型质量，并免去调整容量因子的需要。然而，由于分配给每个专家的令牌数量不固定，底层系统必须高效管理动态计算和内存需求。因此，近期文献主要关注如何针对这些动态且长度可变的工作负载优化计算（Gale 等人，2023年；A minabadi 等人，2025年）。

然而，无论是令牌丢弃还是无丢弃路由方案，都仍面临一个根本挑战：当前实现需要存储索引和压缩后的令牌数据，导致内存占用与$L \times K \times d$成正比。在现代大语言模型训练中，随着序列长度增加、批次大小提高，内存缓冲区会急剧扩张。

示例： 为了说明与令牌分发相关的激活占用，我们以一个真实世界中的混合专家模型（例如 DeepSeek）为例，在此进行定量研究。对于 DeepSeek 模型中的典型混合专家层，它包含 $L \approx 2$ 百万个令牌、 $K = 4$ 个活跃专家，模型维度为 $d = 6144$,

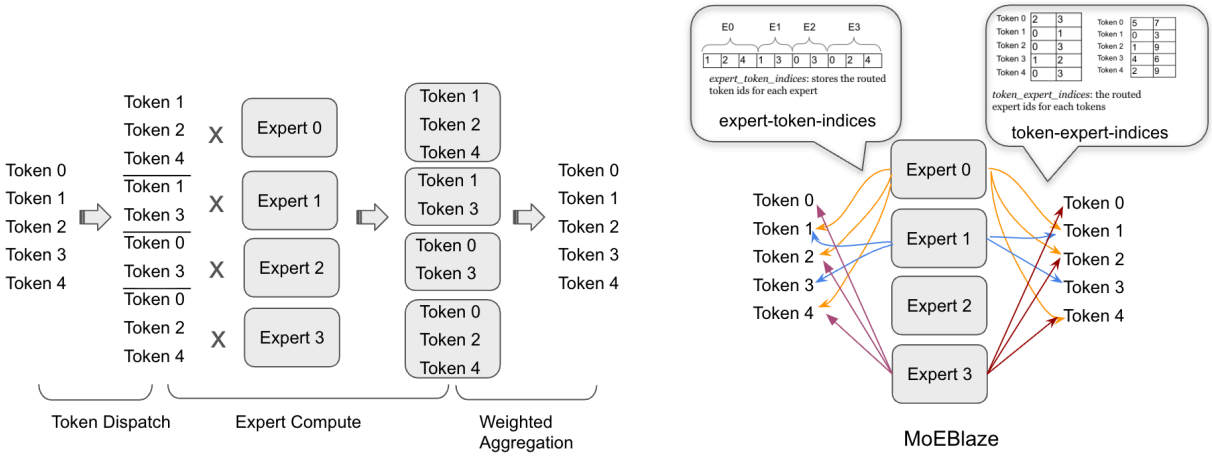


图 *1.* 传统方法中的 MoE 与 MoEBlaze。左图展示了传统的 MoE 计算，包括令牌分发、专家计算和加权聚合（详情见第 2 节）。右图展示了所提出的 MoEBlaze 框架，该框架引入了节省内存的令牌路由和专家计算（详情见第 3 节）。

并且路由令牌缓冲区中每个元素使用 2 个字节（bfloat16），其内存占用为：

$$\text{Mem}_{routing} = L \times d \times k \times 2bytes \approx 94\text{GB}$$

我们可以看到，仅一个路由缓冲区就可能占用近百 GB 内存。

2.2 专家前馈网络（FFN）

令牌分发之后，是各个专家的前馈网络（FFN）计算。每个 FFN 通常由一个两层多层感知机（MLP）实现。第一层将输入从维度 d 投影到更高维的隐藏空间 h，第二层再投影回输出维度 d'（假设 $d' = d$）。所有 E个专家的参数总共需要 $\mathcal{O}(E \times (h \times d + d \times h))$ 内存，但条件计算范式确保每个令牌仅激活 k 个专家，从而使每次前向传播的计算成本保持较低。给定输入 $\mathbf{x}$ 时，专家 E_i 内部的 FFN 计算定义如下：

$$E_i(\mathbf{x}) = W_{2,i} \cdot \sigma(W_{1,i}\mathbf{x})$$

其中 $W_{1,i} \in \mathbb{R}^{h \times d}$ 和 $W_{2,i} \in \mathbb{R}^{d \times h}$，而 σ 是非线性激活函数（例如 ReLU、GELU、SwiGLU）。

第二个主要的内存瓶颈源于 FFN 计算过程中的中间激活值存储。对于一个活跃专家，第一次线性变换 $W_{1,\ i}\mathbf{x}$ 会生成大小为 $L_i \times h$ 的中间激活值，其中 L_i 是路由到专家 i 的令牌数量。虽然活跃专家的数量较少，但前向传播期间，所有 E 个专家的这些激活值所占用的内存总量为 $\mathcal{O}(L \times h)$；而在反向传播期间，由于需要存储中间值以计算梯度，内存占用可能高得多

选择的激活函数（例如 SwiGLU）会进一步加剧内存压力。

示例： 我们以 DeepSeek 的配置为例，说明前馈神经网络计算所产生的激活内存占用为何重要。设有 $L \approx 2$ 百万个输入令牌，前馈神经网络隐藏维度为 $d = 24576$，并且每个元素占用 2 字节（bfloat16），即可得出中间层的激活内存占用：

$$\text{Mem}_{act} = 2L \times h \approx 98\text{GB}$$

2.3 输出聚合

最后一个阶段是输出聚合，即通过加权求和将所选专家的输出合并，为每个令牌生成最终输出。权重由门控网络的分数决定。对于输入令牌$\mathbf{x}$，混合专家的输出 $\mathbf{y}$ 为：

$$\mathbf{y} = \sum_{i=1}^{E} g_i(\mathbf{x}) \cdot E_i(\mathbf{x})$$

其中 $g_i(\mathbf{x})$ 是专家 i的门控分数，只有得分排名前 k 的专家得分非零。聚合输出所需的内存为 $\mathcal{O}(L \times d)$。在计算方面，这涉及 $\mathcal{O}(L \times k \times d)$ 次运算；考虑到稀疏性（$k \ll E$），通常效率较高。

3 内存高效的令牌路由与训练算法

如第 2.1 节所述，在令牌选择型混合专家中，门控网络会将每个输入令牌分配给一个或多个专家。为了

便于在专家计算期间高效地索引和组织令牌，传统系统会将这些令牌紧凑地存入每专家缓冲区。随后执行每专家多层感知机（MLP）模块；在此之前的紧凑化步骤会使中间结果保持为紧凑化后的令牌长度，然后在输出时对其求和并缩减至原始令牌长度。至关重要的是，这种分离和中间存储会在整个 MoE 训练过程中产生大量激活缓冲区。在本节中，我们提出了一种内存高效的路由与专家计算算法，可大幅减少辅助激活内存占用，同时实现高效的 MoE 训练。

给定形状为（$L,\ d$）的激活张量，我们算法的核心思想是利用在令牌分发步骤中生成的辅助索引列表，跟踪路由决策，并在整个混合专家（MoE）计算过程中即时访问令牌并归约结果。具体而言，我们的融合内核按以下方式运行：1. 接收门控决策，并构建专家-令牌索引列表及其他相关索引结构；2. 根据专家-令牌索引列表的指引，从原始、未经置换的激活张量中即时收集数据，执行专家多层感知机计算；3. 专家求和随后使用令牌-专家索引列表，将多层感知机结果直接求和并归约至最终输出张量。通过直接访问输入并仅存储最终结果，我们消除了许多其他论文中为实现物化令牌路由而通常需要的中间激活。令牌-专家索引列表只存储令牌和专家 ID，因此极为轻量。此外，这种方法使我们能够将令牌/专家索引与计算紧密融合，从而为重叠内存访问与计算创造了可能。这在最新的 H100 GPU 等现代硬件上尤其有利，能够提高资源利用率并加快运行速度。

下面我们将详细介绍所提出方法的前向和反向传播。下一节将说明数据结构以及高效构建这些数据结构的方法。

3.1 前向传播

令牌分发： 在令牌分发步骤中，我们不会为路由词元创建专用缓冲区。相反，我们根据前一门控阶段生成的门控分数，构建若干轻量级索引数据结构。这些结构包括：per-expert token list，用于跟踪分配给每个专家的词元 ID；以及每令牌专家列表，用于存储为每个词元选定的专家 ID。在此阶段，不会为物化后的路由词元激活值分配或保留内存。

专家计算： 我们利用 per-expert token list 中记录的索引，从原始未重排的激活张量中即时收集数据，以执行专家计算多层感知机。为最大限度地提高内存效率，在反向传播时仅缓冲两个连续多层感知机之间的中间结果（即第一个多层感知机的输出）。

输出聚合： 专家的最终结果经过聚合，生成最终的 (L, d) 输出。由于我们不会存储物化的令牌分发结果的激活缓冲区，因此该求和操作与第二次多层感知机计算紧密融合，并直接利用每令牌专家列表即时归约至输出张量。

3.2 反向传播

反向传播接收 (L, d) 令牌的梯度，并通过前向步骤的逆过程将其反向传播。专家求和的传统反向传播过程依赖路由令牌激活缓冲区，将 (L, d) 梯度“扩展”或物化为 ($L \times k, d$)“路由梯度令牌”，然后再通过多层感知机专家进行反向传播。然而，我们提出的方法通过使用相同的反向映射索引，避免了这一中间扩展步骤。

1. **专家求和反向传播：** 利用从分发元数据中推导出的令牌映射结构，将（$L,\ d$）梯度张量映射回（$L \times k,\ d$）路由梯度令牌。此过程通过一种高效操作完成：将输出梯度“散射”到物化的中间多层感知机结果张量中的相应位置。

2. **专家计算反向传播：** 接下来，梯度通过多层感知机反向传播。在计算权重梯度时，会用到先前检查点保存的两个连续多层感知机之间的中间结果。

3. **令牌梯度累积：** 最后，根据所有专家的结果累积输入令牌的梯度。此步骤将每个令牌被路由到的k个专家的贡献相加，生成输入激活的最终（$L,\ d$）梯度张量。由于我们没有存储已物化的路由令牌结果的激活值，因此还利用令牌索引数据结构执行即时归约。

4 高效且可并行化的分发与数据结构

4.1 数据结构

我们定义了上述混合专家训练算法所需的关键数据结构，以实现高效的内存使用。

- 专家令牌索引：一个紧凑张量，存储分配给各专家的令牌索引，并将所有专家的索引拼接在一起。在令牌选择式 MoE 训练中，每个令牌选择 k 个专家，因此专家令牌索引的大小为 $L \times k$。此列表对于专家检索分配给自己的输入令牌至关重要。

- 专家令牌偏移量：一个长度为 $E + 1$的数组，用于存储每个专家令牌数量的独占前缀和。对于专家 i，其所分配令牌的索引范围从专家令牌偏移量$[i]$到专家令牌偏移量$[i+1] - 1$。

- 令牌专家索引：令牌专家索引基本上是专家令牌索引的逆映射。它存储每个令牌所路由到的专家 ID，并按令牌 ID 排序。其形状同样为 $L \times k$。在按专家处理令牌时，此列表用于对中间物化结果（例如两个连续的多层感知机之间）进行合并索引。

- 令牌索引映射：一种 $L \times k$紧凑张量，用于存储路由到专家令牌索引列表中的令牌位置。它按原始令牌 ID$i \in L$在逻辑上分组，使令牌能够高效地从中间缓冲区中查找并收集其 k 专家输出，以完成最终的组合步骤。

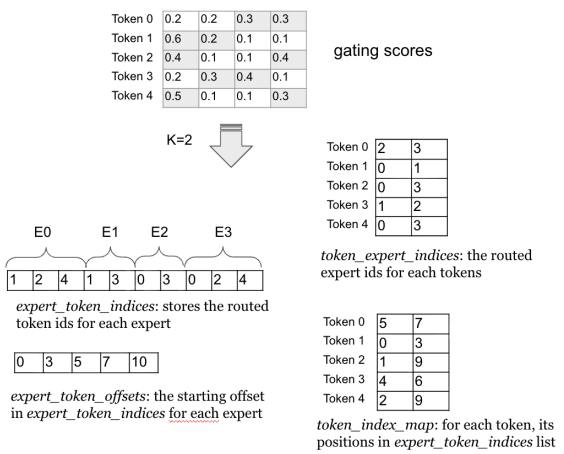


图 2。用于高效节省内存的 MoE 训练的数据结构。

示例。图2 展示了一个示例中 L=6 个令牌、 E=4 个专家和 $k = 2$个激活专家的数据结构。根据门控得分矩阵，我们得到每个令牌的分配结果如下：

令牌 0：专家$\{2, 3\}$；令牌 1：专家$\{0, 1\}$；令牌 2：专家$\{0, 3\}$；令牌 3：专家$\{1, 2\}$；令牌 4：专家$\{0, 3\}$。将令牌的分配结果拼接起来，即可得到令牌专家索引：

$$\text{token_expert_indices} = [2, 3, 0, 1, 1, 2, 0, 3],$$

同样，我们可以得到每个专家对应的路由词元：专家 0：令牌$\{1, 2, 4\}$；专家 1：令牌$\{1, 3\}$；专 家 2：令牌$\{0, 3\}$；专家 3：令牌$\{0, 2, 4\}$。将它们拼接起来，即可得到专家令牌索引和专家令牌偏移量

$$\text{expert_token_indices} = [1, 2, 4, 1, 3, 0, 3, 0, 2, 4],$$
$$\text{expert_token_offsets} = [0, 3, 5, 7, 10]$$

令牌索引映射存储每个令牌在拼接后的专家令牌列表中的位置。例如，令牌索引映射$[0] = \{5, 7\}$，因为令牌 0 被路由至 2 个专家（$k = 2$），并位于专家令牌索引中的第 5 和第 7 个位置。

4.2 高效调度数据结构构建

下面详细介绍如何高效构建上述数据结构。构建过程面临一项挑战：以专家为中心的数据结构设计本身要求采用多对一映射，即将多个令牌分配给同一专家。若采用朴素方法，会导致 GPU 架构上严重的线程级写入争用，从而损害性能。

一种解决方案是采用基于排序的方法来构建令牌分发。该方法将所有令牌的 top-k 个选择结果（topk 专家）展平为一个长度为Lk 的一维数组，其中包含（专家 ID、令牌 ID）元组。随后，按专家 ID 对该数组进行全局排序，以对令牌分组，再通过索引恢复重建令牌顺序，并计算每个专家对应的范围。

这一排序过程在概念上虽然简单，但在大规模场景下会造成严重的性能瓶颈。该排序通过图形处理器上的多遍基数排序实现，其所需的全局内存遍历次数与键宽成正比，导致频繁遍历全局内存，并多次移动 O（Lk）数据。这使得图形处理器上的实际复杂度较高，资源利用率也较低。此外，这一全局排序步骤限制了细粒度并行性，并迫使系统采用多内核调度流水线（多遍排序、分段扫描、索引恢复等），带来较高的

内核启动延迟。这些局限促使我们寻求一种更高效、更适合 GPU 的方法。

为此，我们提出一种高效方法，以可并行化、基于局部索引构建且适合 GPU 的方式，取代复杂的全局排序与组织过程。该方法仅包含三个简单步骤，每一步都经过设计，无需原子操作且可在 GPU 上并行执行，从而最大限度地减少开销高昂的全局内存遍历，并避免复杂的多内核流水线。下面将详细介绍这三个步骤。

构建稠密令牌-专家映射第一步，我们构建一个稠密位图，称为稠密令牌映射，用于编码 top-k 令牌到专家路由。对于每个令牌i，我们考察其 top-k 个分配专家 $\{e_{i,0}, \ldots, e_{i,k-1}\}$。对于每个门控槽位，我们将稠密令牌映射$[i, e_{i,k}]$ 设为 i。其余所有条目均保持未设置状态。

该编码映射的构建过程可在 GPU 上高度并行化。我们首先分配一个$\mathbf{L} \times \mathbf{E}$ 稠密映射，并在 CTA网格上启动内核。我们将每个线程束分配给一个互不重叠的令牌行块（i），并由其加载**top k**专家结果，以此实现并行处理。每个（i, e）对最多只会写入一次，因为每个令牌对应的专家 ID 都是唯一的；这确保不会发生线程束内冲突。

计算专家长度利用已构建的稠密令牌映射，下一步是高效计算每个专家对应的稀疏令牌ID列表的长度和偏移量。我们启动一个自定义内核，将CTA网格映射到稠密令牌映射的列（专家）上。每个CTA专门处理一个专家 e_i，并统计该列中的非零项（令牌到专家的分配）。通过线程束级归约聚合CTA内各行的和，得到专家长度数组。专家长度$[e_i]$ 表示路由到专家 e_i的令牌总数。完成长度计算后，在初始计数内核之外，对专家长度数组执行前缀和，以得出专家偏移量。

将索引路由至门控单元 这第 3 步涉及生成每个专家的令牌 ID 列表，即专家令牌索引，-该列表将作为后续多层感知机计算的输入。为了在 GPU 上以无竞争的方式实现紧凑的、按专家拼接的索引，我们采用了一个围绕生成位置映射展开的两阶段流程。该映射为专家令牌索引列表中密集令牌映射内的每个非零条目指定最终的目标位置ID。建立位置映射后，一个简单的并行内核从密集令牌映射中读取元素，并将其直接写入专家令牌索引中计算得到的对应位置，从而在无需原子操作的情况下实现完全并行.

位置映射的构建可能颇具挑战。我们采用两步策略，确保其构建过程无需原子操作：（i）瓦片级扫描：每个专家启动一个CTA。同一CTA中的线程处理稠密令牌映射中分配给该专家的连续令牌。它们首先在共享内存中计算瓦片级计数，随后在CTA内部局部执行独占扫描（前缀和）操作。（ii）然后，将得到的CTA局部独占扫描计数与预先计算的该专家全局专家偏移量相加。通过这一步相加，即可得到拼接索引数组中正确的最终位置ID。

5 训练–内核协同设计：实现端到端效率

本节详细介绍我们如何联合优化混合专家训练内核与智能激活检查点方法，以解决与某些高级激活方法相关的内存问题。

5.1 SwiGLU 混合专家与内存瓶颈

现代混合专家训练越来越多地采用高级非线性激活函数，例如 *SiLU* 和 *SwiGLU*，以替代 ReLU/GELU。以往研究表明，这些激活函数能够提供更平滑的非线性特性，从而改善优化稳定性，并在大规模语言任务上实现更高的实证准确率。虽然这些激活函数在数值表现上更具优势，但它们也会在训练过程中引入更复杂的计算和更大的内存占用。我们以SwiGLU激活函数为例。SwiGLU激活函数定义如下:

$$\text{SwiGLU}(\mathbf{x}; W_1, W_2) \;=\; \text{SiLU}(\mathbf{x}W_1) \;\cdot\; (\mathbf{x}W_2),$$

其中，SiLU(u) $= u \cdot \sigma(u)$， σ 是 *sigmoid* 运算。对于包含 E 个专家的混合专家层，其中每个专家都实现一个 SwiGLU 前馈网络（FFN），针对单个专家的路由后的令牌批次 $x \in \mathbb{R}^{L \times d}$ 会产生两个投影：
$a = xW_1 \in \mathbb{R}^{L \times h}$ 和 $b = xW_2 \in \mathbb{R}^{L \times h}$。随后执行逐元素运算 SiLU($a$)，以及最终的乘积 SiLU($a$) $\odot\, b$。

5.2 激活检查点与内核协同设计

在传统内核中，前向计算流程必须物化多个中间结果，以便支持反向计算（例如，在 SwiGLU 示例中，包括两个 GEMM 输出 a 和 b、sigmoidσ(a)、SiLU(a) 以及最终乘积）。这些中间结果会被写入全局内存，随后再从中读取，从而产生不可忽略的开销。随着模型和批次大小不断扩展，这会带来大量内存流量和存储成本，并很快成为一个不可忽略的瓶颈。

为缓解观察到的内存压力，我们对混合专家训练流程及其底层 GPU内核进行了联合优化，在不牺牲性能的情况下减少激活内存占用和内存流量。我们的优化基于以下观察：

- 在现代 GPU 上，激活函数的计算通常受内存带宽限制，主要有两个原因：1）激活函数的计算大多是逐点运算，而现代 GPU 非常擅长处理这类运算；2）在大语言模型训练中，我们通常处理令牌数量远大于嵌入维度 $L \gg d$ 的情况。对于这种高而窄形状的矩阵，GPU 上的运算通常受内存带宽限制。

- 虽然激活计算的计算量较小，但其内存占用却出人意料地大。对于复杂的现代激活函数，这一点尤为明显，因为反向传播需要物化并保存许多逐点运算的中间结果。因此，内存分配量相当可观，并随批量大小、序列长度和前馈网络维度线性增长。在当今万亿词元训练环境下，存储这些激活值所需的内存可能大到难以承受。

基于这些观察，我们提出了激活检查点与内核融合相结合的方法。该方法将 SwiGLU 中的两个第一层投影与激活后处理融合，并在专用路径中应用激活检查点，以“打破”由中间激活存储造成的内存墙。

为减少激活值流量和内核启动开销，我们将两个第一层投影和 SwiGLU 后处理融合到单个内核中。该内核接收未物化的路由词元，仅加载一次输入 x，同时将其流经两个（W_1，W_2）通用矩阵乘法，使用寄存器/共享内存计算 SiLU(a)，并立即与 b 相乘，只将最终输出写入全局内存。

这种“后处理融合”消除了 a，b 的全局内存写入，以及后续逐元素运算中的重新读取，从而在可能的情况下，有效地将计算从内存受限领域转移到计算受限领域。与每个投影分别使用独立内核相比，它还将 x 的输入读取量减半。

在反向传播过程中，融合两个第一层投影意味着必须汇总两条路径相对于共享输入 x 的梯度。我们的实现并非分配两个独立的激活缓冲区再将其拼接，而是以融合方式计算两个分支的激活导数，并通过分块归约就地汇总梯度，从而完全消除临时全局缓冲区。

在融合的基础上，我们进一步采用了激活检查点策略——在前向传播期间跳过对 SwiGLU 中间结果（SiLU）的保存。相反，我们在反向传播期间采用重计算策略，利用 SiLU 函数计算开销较低这一特点（例如，它是逐元素操作），而现代 GPU 又严重受限于内存带宽。

算法 1融合 SwiGLU MoE 训练

输入：输入词元 $X \in \mathbb{R}^{L\times d}$，投影权重$W_1$，$W_2 \in \mathbb{R}^{d\times h}, W_3 \in \mathbb{R}^{h\times d}$，输出：输出 $Y_{\text{out}} \in \mathbb{R}^{L\times d}$，梯度$\nabla W_1, \nabla W_2, \nabla W_3, \nabla Z$ *// SwiGLU MoE* 训练的前向模块 *2:* 过程 融合前向传播 *(X, W_1, W_2, W_3) 3: //* 仅加载一次输入词元 *4:* 加载 X *5: //* 第 *1* 个多层感知机投影： *6: //* 计算 A 和 B； *7: //* 在内核中计算 *SiLU(A)* 和 Y_{swi} *8: // SiLU(A)* 是临时结果 *9: (A, B)* $Y_{\text{swi}} \leftarrow$ 融合 *SwiGLU(Z, W_1, W_2) 10:* $Y_{\text{out}} \leftarrow Y_{\text{swi}} W_3$ *11:* 存储 *A, B,* Y_{swi} *12: //* 存储激活值和 *SwiGLU* 输出，供反向传播使用 *13:* 返回 Y_{out} *14: 15:* 过程 融合反向传播 *16: (*$Y_{\text{out}}, \nabla Y_{\text{out}}, W_1, W_2, W_3, A, B, Y_{\text{swi}}$*) 17: //* 最终投影的梯度 *18:* $\nabla W_3 \leftarrow Y_{\text{swi}}^T \nabla Y_{\text{out}}$ *19: //* 将梯度反向传播至 *SwiGLU* 输出 *20:* $\nabla Y_{\text{swi}} \leftarrow \nabla Y_{\text{out}} W_3^T$ *21: //* 加载已存储的激活值 *22:* 加载 *A, B 23: //* 重新计算 *SiLU(A)* 以节省内存 *24:* $S_{\text{recomp}} \leftarrow$*SiLU(A) 25: //* 关于 A的导数 *26:* $\nabla A \leftarrow \nabla Y_{\text{swi}} \odot B \odot \nabla$*SiLU(A) 27: //* 关于 B的导数 *28:* $\nabla B \leftarrow \nabla Y_{\text{swi}} \odot S_{\text{recomp}}$ *29:* $\nabla W_1, \nabla W_2 \leftarrow$ 融合权重反向传播*($X, \nabla A, \nabla B$) 30:* $\nabla X \leftarrow$ 融合输入反向传播*($\nabla A W_1^T, \nabla B W_2^T$) 31:* 返回 $\nabla W_1, \nabla W_2, \nabla W_3, \nabla Z$

5.3 整合实践：SwiGLU MoE 上的端到端训练

算法 1 概述了采用 SwiGLU 激活函数的混合专家模型的端到端训练流程。该伪代码具体展示了第

表 *1*。 实验中使用的 MoE 配置。前馈神经网络隐藏维度设置为输入维度的四倍（前馈神经网络隐藏层大小 _ = _ $4\times$ _ 输入维度）。

	输入维度	专家数量	K	批次	序列长度
CONF1	512	4	1	32	2048
CONF2	1024	8	2	32	2048
CONF3	1024	16	4	32	2048
CONF4	2048	16	4	32	1024
CONF5	512	16	4	32	1024
CONF6	1024	16	4	16	1024
CONF7	2048	8	4	16	512

节5中详述的激活检查点与内核融合的集成方式。为简洁起见，省略了融合内核的底层实现，但其高层方法源自第3节

6 实验

在本节中，我们将 MoEBlaze 与当前最先进的稀疏训练系统 Megablocks **??**进行基准测试，展示其在一系列具有代表性的混合专家（MoE）配置中，在训练速度和内存效率方面均有显著提升。

6.1 实验设置

我们所有实验均在单个 NVIDIA H100 Tensor Core GPU 上进行。软件栈使用 PyTorch 2.0.1 和 CUDA 12.1。我们测量单个混合专家层的端到端训练时间，重点关注稀疏到稀疏计算阶段。我们使用两种不同的激活函数评估性能：ReLU（修正线性单元）和 SwiGLU（Swish门控线性单元）。

我们选取了七种具有代表性的混合专家配置，涵盖输入隐藏维度（d）、专家数量（E）、路由至专家的 top-k 个令牌数，以及常见训练参数（序列长度 L和批量大小 B）的不同取值。具体配置模拟了大语言模型中的常见设置，详见表 1。

6.2 基线与度量指标

我们主要以 Megablocks 作为对比基线。Megablocks 通过自定义内核和高效的令牌分发来优化混合专家训练，是高性能稀疏训练的行业标准。

我们使用两个关键指标评估性能：(1) 训练速度：以 MoEBlaze 相对于 Megablocks 在单次端到端训练过程中的加速比来衡量。训练时间包括前向和反向传播过程，但不包括优化器更新，因为优化器与两种方法的设计无关。数值越高，性能越好。(2) 激活内存占用：以为给定输入保存中间激活张量所分配的总内存量来衡量。为测量激活内存，我们使用 PyTorch 的保存张量钩子
用于追踪并计算在给定输入配置下模型训练过程中分配的确切激活空间。

6.3 使用 SiLU 的 MoE 训练内存效率

如图 3所示，与 Megablocks 基线相比，MoEBlaze 在所有测试配置中始终能显著降低激活内存占用。

MoEBlaze 在输入维度较大、专家数量较多的配置（如 conf4）中，节省的内存尤为显著。具体而言，MoEBlaze 仅需 6,100 MB 内存；与 Megablocks 消耗的 22,000 MB 相比，减少了近 $3.6\times$。对于较小的配置（例如 conf1），激活内存节省则不那么明显；这是预期之中的，因为节省量与序列长度 L 及激活专家数量 k成正比，而这两项在 conf1 中都较小（$k=1$）。峰值激活内存的大幅降低，直接得益于两项核心系统创新：(1) 采用更高效的内存高效令牌分发机制，最大限度地减少中间缓冲区分配；(2) 在定制的激活检查点方案中采用智能重计算。

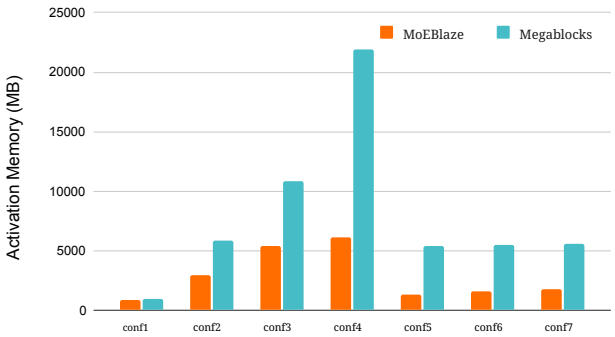


图*3*。 在表 1中定义的各组混合专家配置下，MoE-Blaze 与 Megablocks 使用 SiLU 激活函数时的激活内存占用对比。

6.4 基于 SiLU 的混合专家训练速度

图 4 展示了在给定配置下，MoEBlaze 相较于 Megablocks 的训练加速。MoEBlaze

取得了显著的性能提升，加速倍数为 $1.4\times$ 至 $3.7\times$。

在 conf4（D_{input} =2048，$E = 16$，$L = 1024$，$B = 32$）配置下实现了最高加速比，这表明 MoEBlaze 在模型维度较大时尤其具有良好的扩展性。这些训练速度提升归因于三个因素：(1) 我们高度优化的令牌分发实现，降低了昂贵的令牌分发和置换操作所带来的延迟开销；(2) 高效的数据分发构造内核非常轻量，能够在 GPU 上快速运行，避免了其他基于排序的方法中开销巨大的多遍内核，并大幅消除了 CPU 端瓶颈；(3) 用于批量 GEMM 计算的融合内核能够有效利用 H100 的最新硬件加速特性，例如线程束组矩阵乘法、张量内存加速器等。

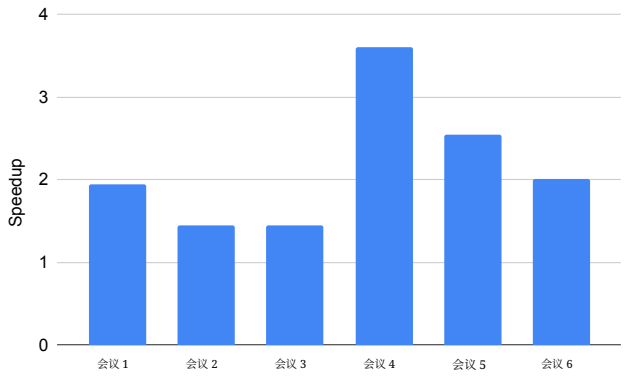


图*4*。在表1 所列配置下，使用 SiLU 作为激活函数时，MoEBlaze 相对于 Megablocks 的加速比。

6.5 使用 SwiGLU 进行混合专家训练的内存效率

由于额外的门控和逐元素乘法操作，SwiGLU 激活函数本身需要占用更多内存。图5展示了在 SwiGLU 设置下的内存消耗对比。与 Megablocks 相比，MoEBlaze 保持了显著的内存优势，峰值激活内存通常不到基线用量的一半。例如，在配置 conf3 中，Megablocks 需要超过 40,000 MB，而 MoEBlaze 仅需约 10,000 MB。这种持续的 $4\times$ 内存压力降低表明，即使面对更复杂的激活函数，我们的内存高效分发和智能重计算方案依然非常有效。

6.6 基于 SwiGLU 的混合专家训练速度 oEs

图4展示了使用 SwiGLU 激活函数时 MoEBlaze 相对于 Megablocks 的加速比。与 ReLU 结果相比，加速比通常更高且更加稳定，范围从 $2\times$ 到 $6.2\times$。相对速度提升源于两个因素：（1）SwiGLU 中更复杂的计算为 MoEBlaze 的高度融合内核创造了更多超越基线的机会；（2）我们的激活优化所节省的内存带宽在 SwiGLU 场景中更为关键，因为其中间激活大小更大，且更

复合起来，从而通过巧妙的内核融合和重新计算减少过多的全局内存访问，使 MoEBlaze 能够更快地执行整个内核。

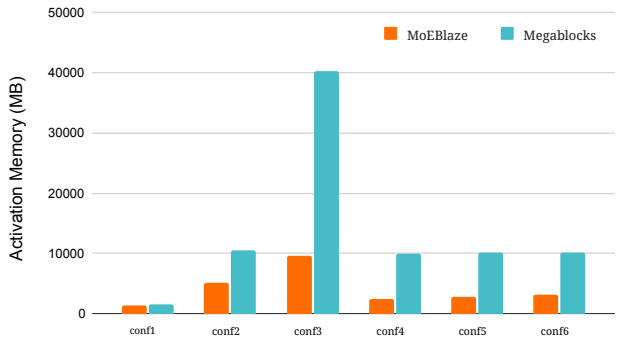


图*5*。 使用SwiGLU 激活函数时，MoEBlaze与Megablocks在表 1所定义的各组混合专家配置下的激活内存占用对比。

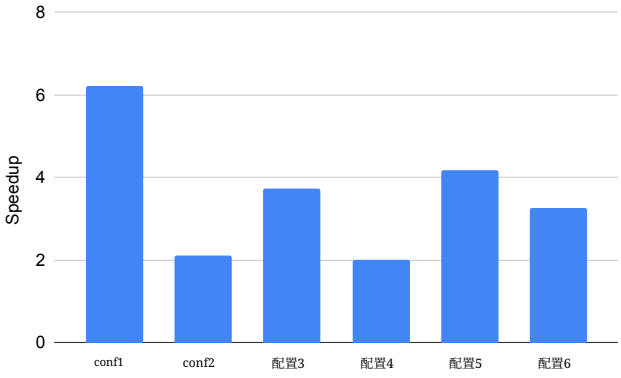


图*6*。 使用SiLU激活函数时，MoEBlaze相较于Megablocks在表 1 所列各组配置下的加速比。

7 相关工作

混合专家架构与扩展。缩放定律和实证研究表明，模型性能会随着计算量、数据集规模和参数数量的增加而可预测地提升（Kaplan 等人，2020年；Hoffmann 等人，2022年）。GPT-3 以 175B 参数实现了少样本泛化（Brown 等人，2020年），后续模型则探索了参数规模、训练数据和上下文长度的扩展（例如，280B 参数的 Gopher（Rae 等人，2022年）；540B 参数的 PaLM（Chowdhery 等人，2023年））。LLaMA 等开放基础模型（Touvron 等人，2023年）通过促进可复现性和广泛评估，加速了研究进展。近来，专有系统提升了多模态整合能力和长上下文能力（例如，GPT-4（OpenAI 等人，2024）、Gemini（Team 等人，2023年）），而开放发布的模型则提炼了此类系统的经验（Gemma（Team 等人，2024））。这些趋势加大了对吞吐量和内存的压力，尤其是在上下文更长、隐藏维度更大的情况下。

混合专家（MoE）最初通过稀疏条件计算在循环架构中扩展神经网络，并凭借稀疏门控混合专家的公式而广受关注（Shazeer 等人，2017）。后续研究在生产级系统中展示了具备自动分片和条件计算能力的大规模基于 Transformer 的混合专家模型（例如 GShard）（Samuel，1959年）。Switch Transformer 将 top-k 专家选择改为 top-1，以简化路由并提升吞吐量（Fedusetal.，2022年）。GLaM 探索了具有专家稀疏性的大规模 MoE 训练，并实现了效率与质量之间的良好权衡（Du etal.，2021年）。在开源生态中，Mixtral 8×7B 采用 top-2 路由，在 32k 上下文长度下表现出色（Jiang 等人，2024），而 DeepSeek-V3 报告了一种拥有 671B 参数、每个 token 激活 37B 参数的 MoE，并实现了高效的大规模训练（DeepSeek-AI etal.，2025年）。

混合专家训练与路由系统。早期以 GPU 为核心的技术栈（如 FastMoE）提供了基于 PyTorch 的分布式混合专家系统，具备实用的加速能力，并支持多节点专家放置（He 等人，2021年）。Tutel 提出了 Flex，这种设计支持运行时自适应并行和流水线，以应对路由造成的工作负载波动，并在不同规模下展现出显著的加速比（Hwang 等人，2023年）。DeepSpeed-MoE 引入了训练和推理优化，以支持大规模的新一代混合专家模型（Rajbhandari 等人，2022年）。Megablocks 将混合专家计算重新表述为块稀疏运算，以避免填充或丢弃令牌，并使其更适用于 GPU（Gale 等人，2023年）。TurboMoE 指出门控路径是核心瓶颈，并引入融合的元数据驱动内核与数据布局转换，以减少稀疏计算开销并提升大规模吞吐量（Aminabadi 等人，2025年）。

路由策略、负载均衡与容量。混合专家模型的质量与效率取决于路由器。稀疏门控混合专家中的辅助负载均衡损失可缓解专家负载不均（Shazeer 等人，2017）；Switch Transformers 采用 top-1 路由和容量约束，以减少计算量并简化收集/散射（Fedus 等人，2022年）。GShard 探索了大规模路由与张量分片策略（Lepikhin 等人，2021年）。Mixtral 等开放式混合专家模型采用 top-2 路由，并调整容量因子以兼顾稳定性和吞吐量（Jiang 等人，2024）。DeepSeek-V3 进一步报告了一种无辅助损失的负载均衡策略，并将训练扩展至极大规模（DeepSeek-AI 等人，2025年）。在这些设计中，路由容量以及令牌丢弃与填充之间的取舍，都会影响吞吐量和内存压力，尤其是在长上下文场景下。

内核与性能优化。MoE 的 GPU 性能取决于数据移动最小化和片上驻留。Megablocks 利用块稀疏内核避免浪费性的稠密填充（Gale 等人，2023年），TurboMoE 则通过定制内核融合门控、散射/收集和专家组合，避免开销高昂的稀疏矩阵乘法（Aminabadi 等人，2025年）。除稀疏映射与编排外，架构感知融合也能缩短激活值生命周期（例如，将 SiLU/SwiGLU 等非线性函数与专家矩阵乘法融合），并减少读写流量。我们的工作 MoEBlaze 沿着这一方向进一步推进，通过紧凑的元数据消除逐专家路由激活缓冲区，并在针对当前一代 GPU 微架构优化的内核中融合路由与专家计算。

8 结论与未来工作

我们介绍了 MoEBlaze，这是一个用于 GPU 上混合专家训练的快速、内存高效系统。MoEBlaze 通过融合令牌分发与计算内核设计，无需物化大型逐专家激活缓冲区。此外，MoEBlaze 整合了混合专家计算与激活流水线，以最大限度地减少读写流量，从而更有效地利用内存带宽并缩减内存占用。我们的实验表明，MoEBlaze 在多种配置下均提供了高效且可扩展的解决方案，峰值激活内存消耗降低超过 $4\times$，端到端训练加速比最高可达 $6.2\times$。

虽然本文主要关注单设备性能，但我们注意到，MoEBlaze 的核心机制同样适用于分布式环境。作为未来工作，我们计划将 MoEBlaze 扩展到分布式训练框架，并研究多节点、多 GPU 混合专家训练的优化方法。

9 致谢

我们衷心感谢 Carole-Jean Wu 提供富有见地的讨论与咨询。我们也感谢 Hong- tao Yu 在 Triton 语言方面提供的专业知识与支持。

参考文献

Aminabadi, R. Y., Holmes, C., Rajbhandari, S., Yao, Z., 和 He, Y. TurboMoE：通过智能内核融合和数据转换增强混合专家模型训练。2025年。

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., 和 Amodei, D. 语言模型是少样本学习者。载于第*34*届神经信息处理系统国际会议论文集，NIPS '20，美国纽约州雷德胡克，2020年。Curran Associates Inc. ISBN 9781713829546。

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pel- lat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. PaLM：利用路径扩展语言建模规模。《机器学习研究杂志》，24(1)，2023年1月。ISSN 1532-4435。

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., 等。《DeepSeek-V3 技术报告》，2025年。网址 https://arxiv.org/abs/2412.19437。

Du, N.、Huang, Y.、Dai, A. M.、Tong, S.、Lepikhin, D.、Xu, Y.、Krikun, M.、Zhou, Y.、Yu, A. W.、Firat, O.、Zoph, B.、Fedus, L.、Bosma, M.、Zhou, Z.、Wang, T.、Wang, Y. E.、Webster, K.、Pellat, M.、Robinson, K.、Meier-Hellstern, K. S.、Duke, T.、Dixon, L.、Zhang, K.、Le, Q. V.、Wu, Y.、Chen, Z. 和 Cui, C. Glam：使用混合专家实现语言模型的高效扩展。载于国际机器学习会议，2021年。网址：
https://api.semanticscholar.org/CorpusID:245124124。

Elfwing, S.、Uchibe, E. 和 Doya, K. 面向强化学习中神经网络函数逼近的 Sigmoid加权线性单元。*CoRR*，abs/1702.03118，2017。网址：http://arxiv.org/abs/1702.03118。Fedus, W.、Zoph, B. 和 Shazeer, N. Switch transformers：利用简单高效的稀疏性扩展至万亿参数模型。《机器学习研究杂志》，23(1)，2022年1月。ISSN 1532-4435。

Gale, T., Narayanan, D., Young, C. 和 Zaharia, M. Megablocks：使用混合专家实现高效稀疏训练。载于 Song, D., Carbin, M. 和 Chen, T.（编），《机器学习与系统会议论文集》，第 5 卷，第 288–304 页。Curan，2023年。

He, J., Qiu, J., Zeng, A., Yang, Z., Zhai, J. 和 Tang, J. FastMoE：一种快速混合专家训练系统。*CoRR*，abs/2103.13262，2021年。网址 https://arxiv.org/abs/2103.13262。

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. 训练计算量最优的大型语言模型。载于神经信息处理系统国际会议第*36*届论文集，NIPS '22，美国纽约州雷德胡克，2022年。Curran Associates Inc. ISBN 9781713871088。

Hwang, C., Cui, W., Xiong, Y., Yang, Z., Liu, Z., Hu, H., Wang, Z., Salas, R., Jose, J., Ram, P., Chau, H., Cheng, P., Yang, F., Yang, M., and Xiong, Y. Tutel：大规模自适应混合专家。载于 Song, D., Carbin, M., and Chen, T.（编），《机器学习与系统会议论文集》，第5卷，第269–287页。Curan，2023年。

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de Las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed,W.E. Mixtralof专家. *CoRR*,abs/2401.04088,2024. doi: 10.48550/ARXIV.2401.04088. URLhttps://doi.org/10.48550/arXiv.2401.04088.

Kaplan, J.、McCandlish, S.、Henighan, T.、Brown, T. B.、Chess, B.、Child, R.、Gray, S.、Radford, A.、Wu, J. 和 Amodei, D. 神经语言模型的缩放定律。*CoRR*，abs/2001.08361，2020年。网址：https://arxiv.org/abs/2001.08361。

Lepikhin, D.、Lee, H.、Xu, Y.、Chen, D.、Firat, O.、Huang, Y.、Krikun, M.、Shazeer, N. 和 Chen, Z. GShard：利用条件计算和自动分片扩展巨型模型。载于 第九届国际学习表征会议，*ICLR 2021*，虚拟会议，奥地利，*2021*年*5*月*3*日至*7*日。OpenReview.net，2021年。网址：https://openreview.net/forum?id=qrwe7XHTmYb。

OpenAI,Achiam, J., Adler, S., Agarwal, S., Ahmad, L.,Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt,J., Altman, S.,Anadkat, S., Avila, R., Babuschkin, I.,Balaji, S., Balcom, V., Baltescu, P., Bao, H.,Bavarian, M.,Belgum, J., Bello,I.,Berdine, J., Bernadett-Shapiro,G.,Berner, C., Bogdonoff, L.,Boiko, O., Boyd, M., Brakman,A.-L.,Brockman, G., Brooks,T., Brundage, M., Button,K., Cai, T., Campbell, R., Cann, A., Carey, B.,Carlson,C., Carmichael, R., Chan, B., Chang, C.,Chantzis, F.,Chen,D., Chen,S., Chen, R.,Chen, J.,Chen, M., Chess,B., Cho, C., Chu, C., Chung, H.W., Cummings, D.,Currier, J., Dai, Y., Decareaux, C., Degry, T.,et al. Gpt-4technicalreport, 2024. URL https://arxiv.org/abs/2303.08774.Rae, J.W., Borgeaud, S., Cai, T., Millican, K., Hoff-mann,J.,Song,F.,Aslanides, J., Henderson, S., Ring,R.,Young, S.,Rutherford, E., Hennigan, T., Menick,J., Cassirer, A., Powell, R., van denDriessche, G.,Hendricks, L.A., Rauh, M., Huang, P.-S., Glaese, A.,Welbl,J., Dathathri, S., Huang, S., Uesato, J., Mellor,J., Higgins,I., Creswell, A., McAleese, N., Wu, A.,Elsen, E., Jayakumar, S., Buchatskaya,E., Budden, D.,Sutherland, E., Simonyan, K., Paganini,M., Sifre, L.,Martens, L., Li, X.L., Kuncoro, A., Nematzadeh, A., Gri-bovskaya, E., , etal. Scaling language models: Methods,analysis & insightsfrom training gopher, 2022. URLhttps://arxiv.org/abs/2112.11446.Rajbhandari, S.,Li, C., Yao, Z.,Zhang,M.,Aminabadi,R. Y., Awan, A.A., Rasley, J., and He, Y. DeepSpeed-MoE: Advancingmixture-of-expertsinference and train-ing to power next-generation AI scale. In Chaudhuri,K.,Jegelka, S.,Song, L., Szepesvari, C., Niu,G., and Sabato,S. (eds.),《第*39*届国际机器学习会议论文集》，第162卷，载于机器学习研究论文集，第18332–18346页。PMLR，2022年7月17日至23日。URL
h
t
t
p
s://
proceedings.mlr.press/v162/rajbhandari22a.html.
Ramachandran, P., Zoph, B., and Le, Q. V. 搜索激活函数。*CoRR*，abs/1710.05941，2017。URLhttp://arxiv.org/abs/1710.05941. Samuel, A. L. 使用跳棋游戏进行机器学习的一些研究。*IBM Journal of Research andDevelo*

*p*ment, 3(3):211–229, 1959年。

Shazeer, N. GLU变体改进了Transformer。*CoRR*，abs/2002.05202，2020年。网址：https://arxiv.org/abs/2002.05202。Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E. 和 Dean, J. 规模惊人的大型神经网络：稀疏门控混合专家层。载于第五届国际学习表征会议（*ICLR 2017*），法国土伦，*2017*年*4*月*24*日至*26*日，会议论文集。OpenReview.net，2017年。网址：https://openreview.net/forum?id=B1ckMDqlg。Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K. 等。Gemini：一系列功能强大的多模态模型。*arXiv*预印本 *arXiv:2312.11805*，2023年。Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivi`ere, M., Kale, M.S., Love,J., Tafti, P., Hussenot, L., Sessa, P.G., Chowdhery, A.,Roberts, A., Barua, A., Botev, A., Castro-Ros, A.,Slone,A.,H´eliou, A., Tacchetti, A., Bulanova, A., Paterson, A., Tsai, B. 等。Gemma：基于Gemini研究与技术的开放模型，2024年。网址：https://arxiv.org/abs/2403.08295。Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozi`ere, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E. 和 Lample, G. Llama：开放且高效的基础语言模型。*CoRR*，abs/2302.13971，2023年。doi：10.48550/ARXIV.2302.13971。网址：https://doi.org/10.48550/arXiv.2302.13971。Williams, S., Waterman, A. 和 Patterson, D. Roofline：适用于多核架构的深刻可视化性能模型。*Commun.ACM*，52(4):65–76，2009年4月。ISSN 0001-0782。doi：10.1145/1498765. 1498785。网址：https://doi.org/10.1145/1498765.1498785。Wulf, W. A. 和 McKee, S. A. 撞上内存墙：显而易见之事的影响。*SIGARCH Comput.Archit.News*，23(1):20–24，1995年3月。ISSN 0163-5964。doi：10.1145/216585.216588。网址：https://doi.org/10.1145/216585.216588。