

Large Transformer Model Inference Optimization

原文地址: [Large Transformer Model Inference Optimization](#) 本文为其中文（修改）翻译版本

关键概念:

Efficient Transformers

推理优化 (Inference Optimization)

蒸馏 (Distillation)

量化 (Quantization)

训练后量化 (Post-training Quantization, PTQ)

感知量化训练 (Quantization-Aware Training, QAT)

剪枝 (Pruning)

稀疏性 (Sparsity)

混合专家模型 (Mixture-of-Experts, MoE)

Kernel Improvement

内容目录:

🔥 方法概览: 📈 蒸馏 (Distillation)、⚖️ 量化 (Quantization)、✂️ 剪枝 (Pruning)、🔧 稀疏化 (Sparsity)

⚖️ 量化细节: ✎ 训练后量化 PTQ、🧠 感知量化训练 QAT、📦 Transformer 量化挑战

✂️ 剪枝细节: 🔎 如何剪枝 (How to prune?)、🔄 如何再训练 (How to retrain?)

🔧 稀疏化: 12/34 N:M 稀疏性剪枝、🧩 稀疏化 Transformer、🌐 MoE 架构、🧭 路由策略改进、⚙️ Kernel 优化

🏗 架构优化: 💬 稀疏注意力模式、🔄 循环机制、🗃 内存节省设计、🎯 自适应注意力

如今，大型 Transformer 模型已成为主流，在众多任务中不断刷新最先进 (SoTA) 成果。尽管这些模型极具性能优势，但其训练和推理成本极高，尤其在真实的大规模应用场景中，**高昂的推理开销**（包括延迟与内存）已成为部署的主要瓶颈。⌚ 为什么大型 Transformer 模型的推理如此困难？除了模型参数规模的持续膨胀，推理难度主要源于以下两个核心问题 (Pope et al., 2022)：

- **内存占用极高**: 推理过程中不仅要存储模型权重，还需保留中间状态。
 - 例如，在解码阶段，KV 缓存必须常驻内存。在批大小为 512、上下文长度为 2048 的设置下，KV 缓存总大小可达 3TB，约为模型本身的 3 倍！
 - 此外，注意力机制的计算开销随着输入长度呈 **二次增长**。
- **难以并行化**: 推理通常为自回归 (auto-regressive) 生成，意味着每一步依赖前一步输出，天然难以并行，从而限制了速度。

在本文中，我们将聚焦于多种提升 Transformer **推理效率**的方法，其中包括通用的压缩策略和专为 Transformer 架构设计的优化方案。

方法概览：如何提升推理效率？

优化 Transformer 推理，通常目标包括：

- 🧠 **减少内存消耗**: 降低对显存资源的依赖；
- ⚡ **减少计算开销 (FLOPs)**: 降低算力需求；
- ⏳ **降低推理延迟**: 提高响应速度。

以下是几类有效的策略：

- **并行化策略 (Parallelism)**: 通过智能地划分模型组件或数据（如数据并行、模型并行、张量并行等），可支持在大规模 GPU 集群上训练和推理万亿参数模型。
- **内存卸载 (Memory Offloading)**: 将暂时不需要的数据移动至 CPU 内存，在使用前再调入 GPU，节省显存资源，代价是一定的延迟。

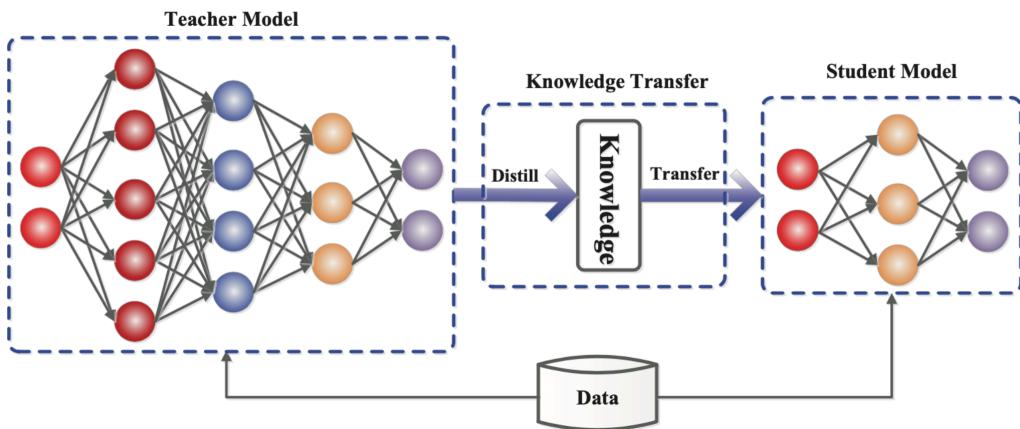
- **智能批处理 (Smart Batching)**: 例如 Effective Transformer 中提出的按序列长度聚类, 从而减少 padding, 提高 batch 内 token 的利用效率。
- **网络压缩 (Network Compression)**: 包括剪枝 (pruning)、量化 (quantization)、蒸馏 (distillation) 等。压缩后的模型通常具备更小的存储需求和更快的运行速度。
- **架构级优化 (Architectural Optimization)**: 特别是针对注意力机制的重构, 如稀疏注意力、低秩分解、自适应路径等, 可极大提升解码效率。

关于如何在大规模 GPU 集群上训练和部署大型模型的并行策略与显存优化方法 (如 CPU offloading), 可参考: [How to Train Really Large Models on Many GPUs?](#)

本文将重点聚焦于推理阶段的网络压缩技术和架构层面的优化策略。

蒸馏 (Distillation)

知识蒸馏 (Knowledge Distillation, KD) ([Hinton et al., 2015](#), [Gou et al., 2020](#)) 是一种经典的模型压缩方法, 通过将大型“教师模型”的能力迁移给一个更小的“学生模型”, 从而在保持性能的同时大幅降低推理成本。蒸馏对学生模型的架构几乎没有限制, 只需在输出空间对齐, 以构造合理的学习目标。其基本流程如下图所示:



学生模型通过最小化其输出与教师输出之间的差异来进行训练。以 softmax 输出为例, 设教师模型的 logits 为 z_t , 学生模型的 logits 为 z_s , 则蒸馏损失通常包括高温 softmax 下的分布对齐, 以及可选的监督学习目标 (如交叉熵)。综合损失函数为:

$$\mathcal{L}_{\text{KD}} = \mathcal{L}_{\text{distill}}(\text{softmax}(z_t, T), \text{softmax}(z_s, T)) + \lambda \mathcal{L}_{\text{CE}}(y, z_s),$$

其中 T 是温度参数, λ 控制监督信号与蒸馏信号的权重。

一个典型案例是 DistilBERT ([Sanh et al., 2019](#)), 它通过蒸馏将 BERT 模型的参数量减少了约 40%, 在保留约 97% 性能的同时, 推理速度提升了 71%。其训练目标整合了三部分损失:

1. **软标签蒸馏损失 (Soft Distillation Loss)**: 使学生模型在预测的概率分布上接近教师模型;
2. **掩码语言建模损失 (Masked Language Modeling, MLM)**: 维持原有的预训练任务;
3. **隐藏状态对齐损失 (Hidden State Alignment Loss)**: 鼓励学生模型的中间层输出与教师模型相应层的隐藏状态保持一致, 通常使用余弦相似度度量两者之间的相似性。

通过隐藏状态对齐, 学生不仅能模仿教师模型的最终预测结果, 还能在“思考路径”上尽可能一致, 从而更好地传承教师模型的能力。

知识蒸馏还可与量化、剪枝、稀疏化等其他压缩技术联合使用: 教师模型通常保持为完整的全精度模型, 而学生模型则可采用压缩后的形式, 从而实现更高的效率与更低的硬件需求。

量化 (Quantization)

在深度神经网络中，**量化 (Quantization)** 是一种常用的压缩方法，主要分为以下两种：

1. **训练后量化 (Post-Training Quantization, PTQ)**：先完成全精度训练，再将模型的权重转换为低精度表示。该方法实现简单、计算开销小。
2. **量化感知训练 (Quantization-Aware Training, QAT)**：在训练或微调过程中引入量化，使模型在训练中适应低精度表达。训练成本较大。

需要注意的是：理论上的量化优化并不总能在实际硬件上取得理想速度收益。某些低精度操作（如 INT4 × FP16）在 GPU 上的硬件支持仍然有限，因此具体效果需结合系统测试。

Transformer 量化的挑战

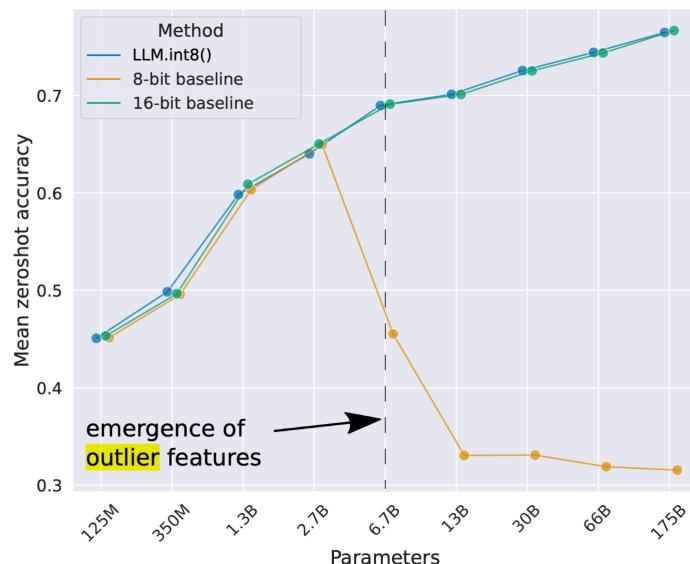
Transformer 模型在进行低精度量化时，尤其容易出现性能下降的情况。这主要是由于：

- **激活值的动态范围过大**：尤其是在 FFN 层前后或残差连接中，传统统一量化方法难以保持模型表达能力。
- **异常激活值 (outliers)**：激活值中存在数量极少但幅值极大的 outlier，严重干扰量化精度，尤其在大模型（如 OPT-6.7B）中普遍存在。

例如，[Bondarenko et al. \(2021\)](#) 发现，在 BERT 的前馈网络（FFN）中，输入输出的动态范围差异极大，导致单一量化误差显著。下标展示了直观的例子：只将模型权重量化为 8 位，同时保持激活为全精度（w8A32），与将激活量化为 8 位（无论权重是否为较低精度）（w8A8 和 w32A8）相比，结果要好得多。

Configuration	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE
FP32	57.27	93.12	88.36	89.09	89.72	84.91	91.58	70.40	83.06
W8A8	54.74	92.55	88.53	81.02	83.81	50.31	52.32	64.98	71.03
W32A8	56.70	92.43	86.98	82.87	84.70	52.80	52.44	53.07	70.25
W8A32	58.63	92.55	88.74	89.05	89.72	84.58	91.43	71.12	83.23

[Dettmers et al. \(2022\)](#) 观察到，在规模超过 6.7B 的 OPT 模型中，各层都会出现高幅值的 outlier 激活值。这些异常值在部分维度上可能比其他值高出 100 倍，严重影响性能。对应下图结果：



训练后量化 (Post-training Quantization, PTQ) 策略

以下介绍 4 种 **训练后量化 (Post-training Quantization, PTQ)** 的策略，包括混合精度量化 (Mixed-Precision Quantization)，精细粒度的量化 (Fine-grained Quantization Granularity)，利用二阶信息的量化方法 (Second order information for quantization) 以及异常值平滑 (Outlier Smoothing)。

混合精度量化 (Mixed-Precision Quantization)

解决上述量化挑战的一个直接方法是对权重与激活值使用不同精度的量化策略。

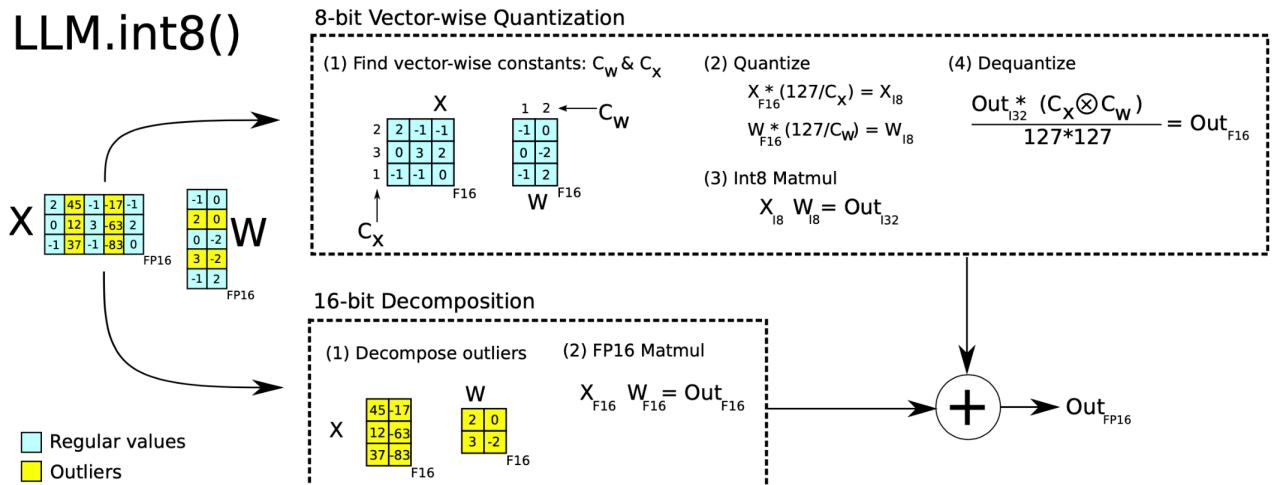
GOBO (Zadeh et al., 2020) 是最早在 Transformer (如小型 BERT) 模型上应用训练后量化的工作之一。该方法假设每层权重呈高斯分布，通过追踪每层的均值和标准差识别异常值 (outliers)。对于这些异常值，保留其原始浮点形式；其余权重则划分为多个区间，只需存储索引和每个区间的中心值 (centroid) 即可。

基于一个关键观察 —— 仅有 BERT 中部分激活层 (如 FFN 后的残差连接) 在量化后会显著影响模型性能，

Bondarenko et al. (2021) 提出了 **混合精度量化策略 (Mixed-Precision Quantization)**：对这些“问题激活”采用 16-bit 表示，其余则使用 8-bit 表示，从而在性能与效率之间取得平衡。

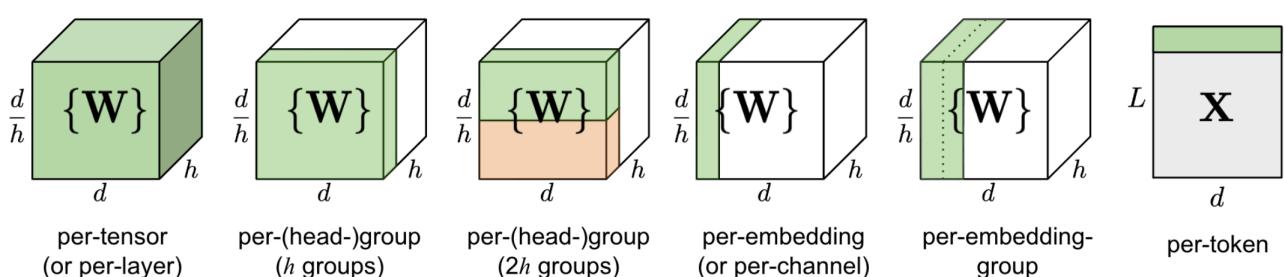
LLM.int8() (Dettmers et al., 2022) 进一步提出两种混合精度的量化方式：

- **按内积量化 (per-dot product quantization)**：将矩阵乘法视为多个独立的内积计算，对每一行和每一列分别使用其绝对最大值进行缩放，然后量化为 INT8。
- **异常激活保留 FP16**：当某些激活值显著大于其他维度 (如超过 20 倍)，则保留这些“异常值”为 FP16，其余维度量化为 INT8。这些 outliers 的识别基于经验规则。



精细粒度的量化 (Fine-grained Quantization Granularity)

下图展示了不同粒度 (Granularity) 下的量化比较，其中 d 表示模型的隐藏维度 (hidden state dimension) 大小， h 是一个多头注意力 (MHSA) 模块中头的数量。



统一对整层权重进行量化（即 *per-tensor* 或 *per-layer* 量化）是实现成本最低的方法，但这种做法在粒度上较粗，通常无法带来理想的效果。

Q-BERT (Shen, Dong & Ye et al., 2020) 对经过微调的 BERT 模型采用了分组量化 (group-wise quantization) 策略：将多头注意力机制中的每个注意力头所对应的权重矩阵视为一个独立的组，并对每个组应用基于 Hessian 矩阵的混合精度量化。

受到类似观察启发，**PEG** (Per-embedding Group) 方法提出了对激活值进行分组量化。研究发现激活值的异常值通常只集中在（隐藏维度/模型大小）中的少数几个维度 d 中 (Bondarenko et al. 2021)。每个维度单独量化开销过大，因此 PEG 方法将激活张量沿嵌入维度划分为多个大小相同的组，每组共享相同的量化参数。为确保异常值尽可能地聚集在同一组中，PEG 会对嵌入维度做一个确定性的排序，按各维度的值域 (range) 对其进行排列。

ZeroQuant (Yao et al., 2022) 在权重量化上同样采用 group-wise 策略，但在激活量化上创新性地引入了 **token-wise quantization**：以 token 为单位分组处理激活张量。此外，为减少频繁的量化/反量化操作带来的开销，ZeroQuant 设计了融合自定义算子的高效实现方式，从而显著提高了部署效率。

利用二阶信息的量化方法 (Second order information for quantization)

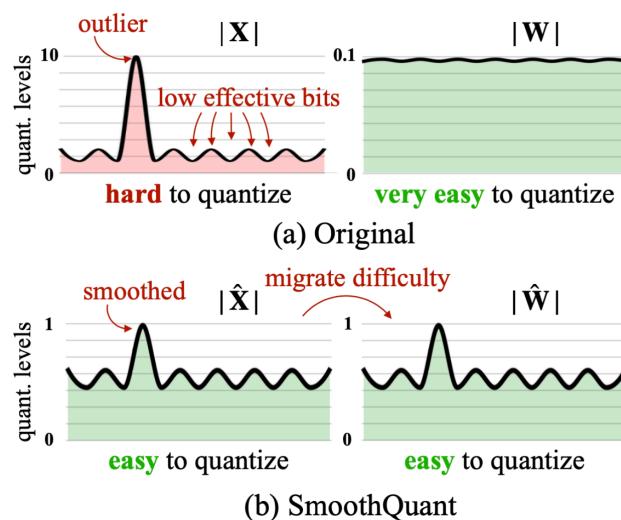
Q-BERT (Shen, Dong & Ye, et al. 2020) 提出了一种基于 Hessian 感知的量化方法 HAWQ (Hessian Aware Quantization)，作为其混合精度量化框架的核心思想。其出发点在于：具有较大 Hessian 谱（即最大特征值较大）的权重参数对量化更为敏感，因此应优先保留更高的表示精度。这种方法也可以视为一种识别并处理“异常值”的机制。从另一个角度来看，权重量化也可被形式化为一个优化问题。给定权重矩阵 W 与输入矩阵 X ，目标是寻找一个量化后的权重矩阵 \hat{W} ，以最小化其引起的输出误差（如均方误差 MSE）

$$\hat{W}^* = \arg \min_{\hat{W}} \|WX - \hat{W}X\|^2$$

GPTQ (Frantar et al. 2022) 将权重矩阵 W 看作由若干行向量 w 组成，对每一行单独进行量化。GPTQ 采用贪心策略，逐步选择对误差影响最大的权重进行量化，并基于 Hessian 矩阵计算出封闭形式的更新公式。。其后续工作 OBQ (Optimal Brain Quantization；Frantar & Alistarh, 2022) 对其进行了进一步扩展和优化。GPTQ 能够将如 OPT-175B 这类模型的权重精度压缩到 3~4 bit，而几乎不会带来性能损失，但该方法仅适用于权重量化，不支持激活量化。

异常值平滑 (Outlier Smoothing)

相比于权重，Transformer 模型中的激活值量化通常面临更高的挑战。为缓解这一问题，**SmoothQuant** (Xiao & Lin, 2022) 提出了一种巧妙的处理方法：通过数学等价变换，将激活中的异常值“转移”到权重中，从而实现激活与权重的联合量化（如 w8A8 方案）。该方法相比混合精度方案具有更高的硬件执行效率。其基本思路如图所示：



SmoothQuant 定义了每通道一个平滑因子 s ，并对权重做缩放：

$$Y = (X \text{diag}(s)^{-1}) \cdot (\text{diag}(s)W) = \hat{X}\hat{W}$$

这个平滑因子可以在模型部署前与前一层的参数一起融合。一个超参数 α 控制了从激活转移至权重的程度： $s = \max(|X_j|)^\alpha / \max(|X_j|)^{1-\alpha}$ 。实验发现，对于许多大型语言模型， $\alpha = 0.5$ 是一个有效的默认值。当模型的激活中包含更多 outliers 时，可以适当调大 α 。

感知量化训练（Quantization-Aware Training, QAT）

感知量化训练（Quantization-Aware Training, QAT） 通过将量化操作显式地纳入预训练或微调过程，使模型能够在训练阶段学习低比特精度下的参数表示，从而显著提升量化模型的最终性能，尽管训练成本相对较高。

一种最直接的 QAT 方式是：在原始或代表性的预训练数据上，对已量化模型进行微调。训练目标可选择原始的预训练任务（如 NLL 或 MLM），也可以是具体下游任务（如分类）的目标函数（如交叉熵）。

另一类更为灵活的方法是基于知识蒸馏，将全精度模型视为教师网络、量化模型视为学生网络，通过蒸馏过程优化学生模型的表现。这类方法往往不依赖原始训练数据——甚至 Wikipedia 数据或随机生成的 token 序列即可取得良好效果。**Layer-by-layer Knowledge Distillation (LKD)** [Yao et al., 2022](#) 进一步提出了逐层量化与蒸馏的策略：每次只量化一个网络层，并以其对应的未量化版本作为教师模型。其损失函数形式如下，对于第 k 层：

$$\mathcal{L}_{\text{LKD},k} = \text{MSE}(L_k \cdot L_{k-1} \cdot L_{k-2} \cdots L_1 - \hat{L}_k \cdot L_{k-1} \cdot L_{k-2} \cdots L_1),$$

其中 L_k 表示第 k 层的原始权重， \hat{L}_k 为其对应的量化版本。

剪枝（Pruning）

剪枝（Pruning） 是一种通过移除模型中不重要的权重或连接，来减少模型规模并保持其表达能力的方法。根据对网络结构的影响，剪枝可分为非结构化剪枝与结构化剪枝。

- **非结构化剪枝（Unstructured pruning）**：可任意移除单个权重或连接，而不保留网络的原始结构。这种方法虽能达到较高稀疏度，但对现代硬件不友好，难以在推理中实现实质加速。
- **结构化剪枝（Structured pruning）**：以保持稠密矩阵乘法的计算模式为目标，按特定模式移除参数块或通道。为了匹配硬件加速器的计算内核，往往需要遵循特定的稀疏结构约束。本文主要关注结构化剪枝，以便在 Transformer 模型中实现高效稀疏化。

剪枝的典型流程包括以下三步：

1. 训练一个稠密网络至收敛；
2. 对网络进行剪枝，移除冗余权重或结构；
3. 可选地对剪枝后的网络进行再训练，以恢复性能。

剪枝的思想与 **彩票假设（Lottery Ticket Hypothesis, LTH）** 密切相关：一个随机初始化的稠密网络中包含多个潜在的“中奖子网络”，其中某些稀疏子网络在单独训练时可达到接近甚至超过完整网络的性能。

如何剪枝（How to Prune?）

幅度剪枝（Magnitude Pruning） 是最简单且有效的剪枝方法，其思路是移除绝对值最小的权重。研究表明（[Gale et al., 2019](#)），这种简单策略可实现与复杂剪枝方法（如变分 dropout ([Molchanov et al., 2017](#)) 和 l_0 正则化 ([Louizos et al., 2017](#))) 相媲美甚至更优的效果。幅度剪枝实现简单，且在不同超参数设置下都能保持较稳健的表现，适用于大规模模型。

Zhu & Gupta (2017) 指出，大而稀疏的模型往往能优于小而稠密的模型。他们提出了 **渐进式幅度剪枝 (Gradual Magnitude Pruning, GMP)**，在训练过程中逐步提升稀疏度。具体做法是在每个阶段剪除绝对值最小的一部分权重（屏蔽为零），并使其在反向传播时不再更新；目标稀疏度随训练逐步增加。GMP 对学习率调度较敏感，需要比稠密模型训练时更高的学习率，但不能高到影响收敛。

迭代剪枝 (Iterative Pruning) (Renda et al., 2020) 则不断交替执行剪枝与再训练：每次仅剪去少部分权重，再微调模型，以此迭代至目标稀疏度。

如何再训练 (How to Retrain?)

剪枝后的再训练可以是简单的 **微调 (fine-tuning)**，利用预训练数据或下游任务数据来恢复性能。

彩票假设 (LTH) 提出了 **权重回滚 (weight rewinding)** 策略：剪枝后，将未被剪掉的权重重置为初始训练时的参数值，然后继续使用原有的学习率调度重新训练。

另一种方法是 **学习率回滚 (Learning Rate Rewinding)** (Renda et al., 2020)：保留剪枝后剩余权重的当前值不变，仅将学习率重置为训练早期的值。他们的实验结果显示：

1. 使用权重回滚的再训练效果普遍优于普通微调；
2. 学习率回滚的表现可与权重回滚持平甚至更好。

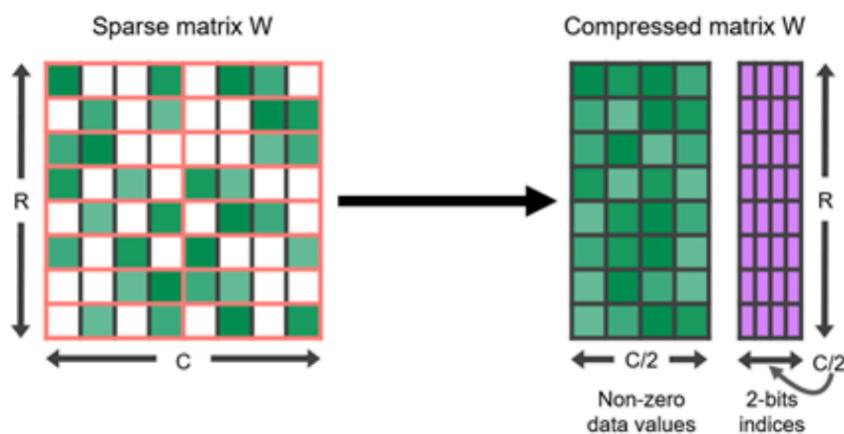
稀疏性 (Sparsity)

稀疏性是一种有效扩展模型容量而不会显著增加推理计算成本的手段。Transformer 中两类主要的稀疏性：

- **稀疏化的稠密层**，如自注意力层和前馈网络 (FFN)；
- **稀疏模型结构**，如引入专家混合 (Mixture-of-Experts, MoE) 模块的架构。

N:M 稀疏性剪枝 (N:M Sparsity via Pruning)

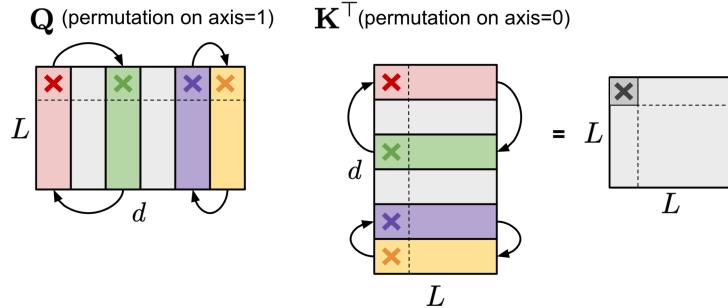
N:M 稀疏性 (Sparsity) 是一种结构化稀疏范式，其中每连续 M 个元素中仅保留 N 个非零值，其余为零。这种剪枝方式不仅利于压缩模型，而且具有良好的硬件友好性。例如，**NVIDIA A100 GPU** 支持 2:4 稀疏模式，可提升稀疏矩阵乘法的推理速度。



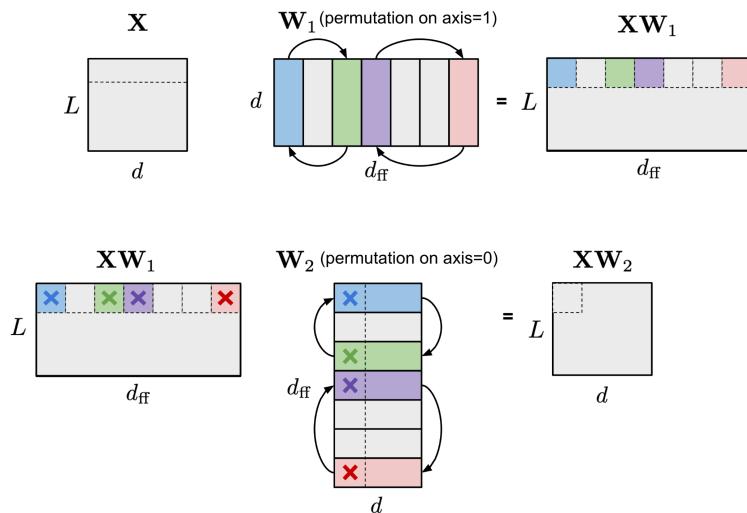
为了将稠密模型转换为符合 N:M 稀疏性要求的结构化模型，推荐的训练流程包括：**全量训练 → 结构化稀疏剪枝 (如满足 2:4 模式) → 微调再训练 (NVIDIA, 2020; Weng, 2023)**。为提升剪枝阶段的保留率，常引入

列置换 (Column Permutation) 操作，将重要权重聚集在每组 M 个参数中，从而更容易满足 N:M 稀疏约束 (如 2:4) 并避免误剪 (Pool & Yu, 2021)。这一操作可在不改变矩阵乘积结果的前提下，提升稀疏结构与硬件加速单元之间的匹配度。具体而言，列置换可在以下常见结构中安全应用：

1. 自注意力机制中的 Query-Key 点积计算：如果对 Query 权重矩阵的列 (axis=1) 和 Key 权重矩阵的行 (axis=0) 应用相同的列置换顺序，则不会改变它们乘积的计算结果。



2. 前馈神经网络 (FFN) 中的线性层对：可以对第一个线性层的权重矩阵进行列置换 (axis=1)，同时对第二个线性层的权重矩阵进行行置换 (axis=0)，使用相同的置换顺序，也不会影响前向传播结果。



为了实现 N:M 结构稀疏性，通常将权重矩阵的列划分为若干由 M 列组成的子组（称为 stripes）。可以证明，在每个 stripe 内重新排列列的顺序，或调换不同 stripe 的位置，并不会影响剪枝后模型对 N:M 模式的结构约束。这为列置换提供了灵活的优化空间。

为充分利用这一自由度，Pool & Yu (2021) 提出了一种迭代贪心列置换算法，旨在在满足 N:M 稀疏约束的前提下，最大化保留的幅值权重。其基本思路为：在每轮迭代中，尝试交换所有可能的通道对，仅保留能带来保留幅值最大增益的那次交换作为新的置换状态。为避免贪心策略陷入局部最优，该方法引入了两种增强机制：

- **有界回归 (Bounded regressions)**：引入一定概率的随机列交换，并限制其最大迭代次数，以实现“广而浅”的搜索过程；
- **窄而深的搜索 (Narrow, deep search)**：选取多个 stripe 进行联合优化，通过局部深入搜索进一步提升全局性能。

Algorithm 1: “Deep” greedy permutation search with bounded regressions to escape local minima

```

def Find_Permutation(matrix, num_cols, stripes_per_group=2, escape_attempts=100):
    permutation = [c for c in range(0, num_cols)]; #Identity permutation
    for escape_attempt in range(escape_attempts+1, 0, -1):
        #Greedy phase: optimize stripe groups that give large benefits
        while True:
            optimizations = Optimize_Stripes_Groups(matrix, permutation, stripes_per_group);
            optimization = Find_Largest_Positive_Improvement(optimizations);
            if optimization is None: break;
            permutation = Permute(permutation, optimization);
        #Escape phase: attempt to escape the local minimum
        if escape_attempt > 1:
            src, dst = Unique_Random_Integers(num_cols);
            permutation = Swap_Columns(permutation, src, dst);
    return permutation; #Final permutation

```

实验证明，在剪枝前先应用列置换策略的模型，剪枝后能显著优于直接对默认通道顺序模型进行剪枝的基线模型，表现为更高的精度保留率和更低的性能损失。

Zhou & Ma, et al. (2021) 扩展了 **STE (Straight-Through Estimator)** (Bengio et al. 2013)，使其能够用于幅度剪枝 (magnitude pruning) 和稀疏参数更新，从而支持从头训练具有 N:M 稀疏性的模型。

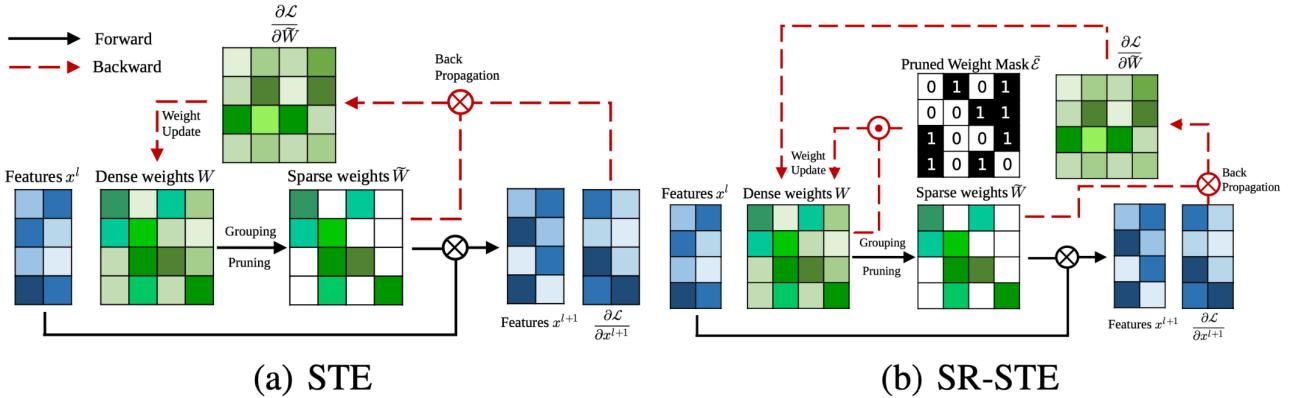
传统 STE 在反向传播中，使用剪枝后网络输出 \widetilde{W} 的梯度 $\partial\mathcal{L}/\partial\widetilde{W}$ ，并将该梯度近似用于更新稠密网络 W ，即：

$$W_{t+1} \leftarrow W_t - \gamma \frac{\partial\mathcal{L}}{\partial\widetilde{W}}$$

为了增强对稀疏结构的控制，作者提出了 **SR-STE (Sparse-Refined STE)**，其在 STE 基础上引入了稀疏掩码引导的修正项：

$$W_{t+1} \leftarrow W_t - \gamma \frac{\partial\mathcal{L}}{\partial\widetilde{W}} + \lambda_W (\bar{\mathcal{E}} \odot W_t)$$

其中 $\bar{\mathcal{E}}$ 是 \widetilde{W} 的二值掩码，表示被剪除的位置。SR-STE 的核心目标是增强训练阶段的掩码稳定性，具体体现在两方面：(1) 抑制 \widetilde{W}_t 被剪除权重的波动，避免“抖动”；(2) 鼓励 \widetilde{W}_t 非剪除权重的进一步优化。直观示意图如下：



与 STE/SR-STE 不同，**Top-KAST** (Jayakumar et al., 2021) 是一种在训练过程中始终保持稀疏结构的优化方法，它无需依赖稠密权重或稠密梯度传播。

Top-KAST 在每一训练步 t 中分为两个阶段：

- **稀疏前向传播 (Sparse forward pass):** 按层选择幅值最大的前 $D\%$ 权重组成子集 $A^t \subset \Theta$ ，其他权重置为 0，形式化来讲

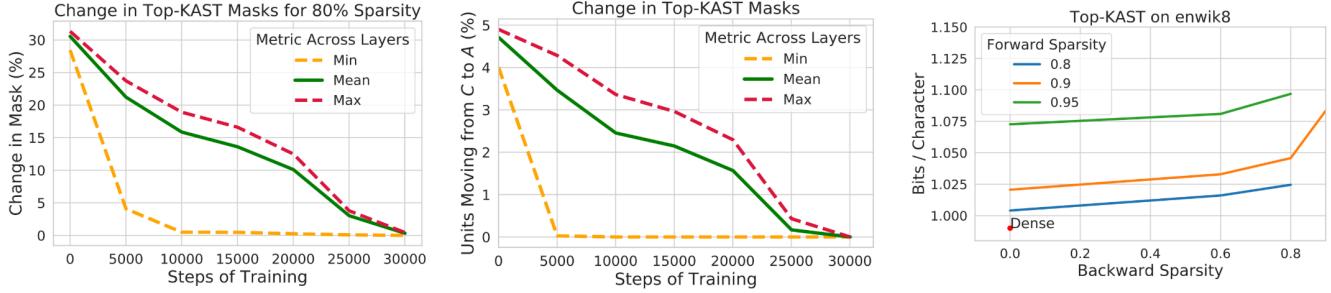
$$\alpha_i^t = \begin{cases} \theta_i^t & \text{if } i \in A^t = \{i | \theta_i^t \in \text{TopK}(\theta^t, D)\} \\ 0 & \text{otherwise} \end{cases}$$

其中 $\text{TopK}(\theta, x)$ 表示基于幅值选择 top x 比例的权重 θ 。

- **稀疏反向传播 (Sparse backward pass):** 使用更大比例 (如 $(D + M)\%$) 的权重子集 $B^t \subset \Theta$ 接收梯度更新，因此 $A^t \subset B^t$ 。这可以增加对掩码变化的探索空间，提高训练阶段 top D -proportion 权重置换的可能性。

$$\Delta_{\theta_i^t} = \begin{cases} -\eta \nabla_{\alpha_i^t} \mathcal{L}(y, x, \alpha^t)_i & \text{if } i \in B^t = \{i | \theta_i^t \in \text{TopK}(\theta^t, D + M)\} \\ 0 & \text{otherwise} \end{cases}$$

训练过程中，Top-KAST 会逐渐固定剪枝掩码，从初期的探索过渡到后期的稳定阶段，如下图所示：



此外，为避免“强者恒强”现象（即已有高幅值权重长期占据激活位置），Top-KAST 通过对当前激活权重引入 L2 正则化，鼓励新的候选参数参与竞争。在更新中，对 B/A 的惩罚强于对 A 的惩罚，从而提高掩码的稳定性：

$$\mathcal{L}_{\text{penalty}}(\alpha_i^t) = \begin{cases} |\theta_i^t| & \text{if } i \in A^t \\ |\theta_i^t|/D & \text{if } i \in B^t/A^t \\ 0 & \text{otherwise} \end{cases}$$

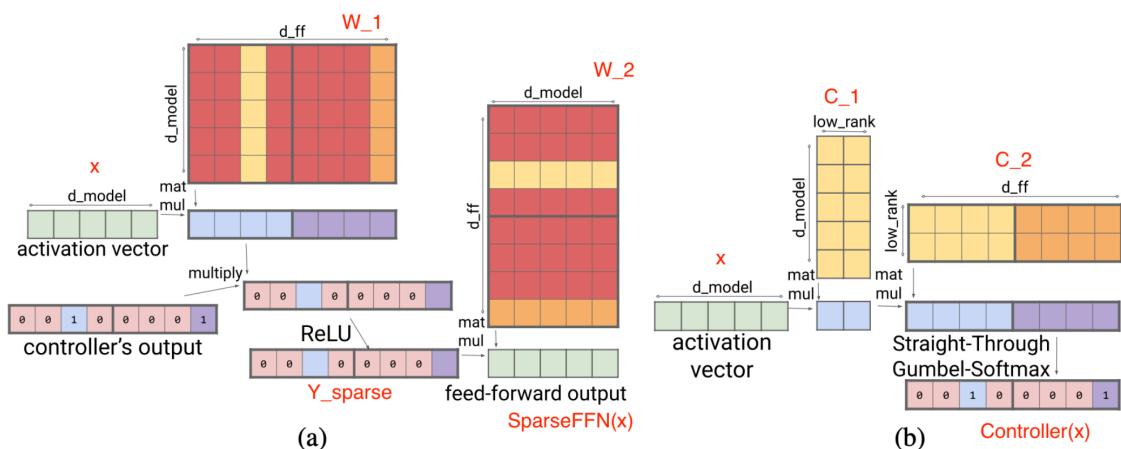
稀疏化的 Transformer (Sparsified Transformer)

Scaling Transformer (Jaszczur et al., 2021) 在自注意力 (Self-Attention) 和前馈神经网络 (FFN) 层中引入稀疏结构，极大提升了推理效率。在单样本 (single-token) 推理任务中，其最高可带来高达 37 倍的速度提升。

稀疏 FFN 层 (Sparse FFN layer)：传统 FFN 层通常由两个全连接层 (MLP) 与一个中间的 ReLU 激活函数组成。在标准实现中，ReLU 会导致大量激活值为 0，而这些值在推理中仍会被参与计算，造成资源浪费。为此，Scaling Transformer 在 ReLU 输出上施加结构化稀疏约束：将中间激活拆分为多个包含 N 个元素的 block，在每个 block 内仅保留一个非零激活，其余置零。形该稀疏模式为**动态选择**，即不同 token 对应的激活位置不同，从而保留表达能力。具体形式如下：

$$\begin{aligned} Y_{\text{sparse}} &= \max(0, xW_1 + b_1) \odot \text{Controller}(x) \\ \text{SparseFFN}(x) &= Y_{\text{sparse}}W_2 + b_2 \\ \text{Controller}(x) &= \arg \max(\text{reshape}(xC_1C_2, (-1, N))) \end{aligned}$$

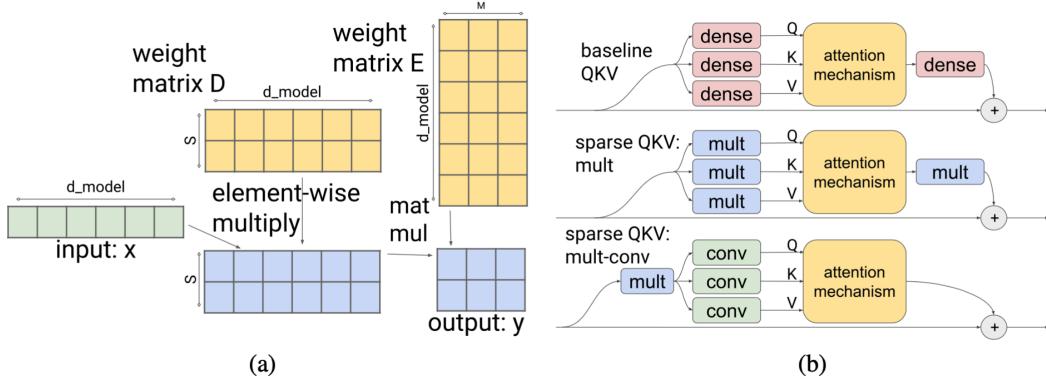
其中每个激活单元与 W_1 的一列和 W_2 的一行一一对应。控制器 $\text{Controller}(x)$ 由一个低秩瓶颈的全连接层 $C_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{lowrank}}}$, $C_2 \in \mathbb{R}^{d_{\text{lowrank}} \times d_{\text{ff}}}$ 实现，其中 $d_{\text{lowrank}} = d_{\text{model}}/N$ 。由于 $\text{Controller}(x)$ 可在加载 FFN 权重之前预先计算，因此我们可在推理时提前获知哪些列将被置零。由此可以在执行矩阵乘法前**跳过这些冗余列的加载与计算**，从而显著加速推理过程。这种稀疏激活 + 列跳过机制是实现高效推理的关键。训练阶段则使用 Gumbel-Softmax (Jang et al. 2016) 近似替代 $\arg \max$ ，实现可导训练过程。示意图如下：



稀疏 QKV 注意力层 (Sparse QKV attention layer): 为稀疏化注意力计算, Scaling Transformer 将注意力层输入维度 d_{model} 被划分为 S 个模块, 每个模块大小为 $M = d_{\text{model}}/S$ 。为了保证每个模块仍能访问嵌入向量的全部信息, 引入了一个乘法层 (multiplicative layer), 该层通过逐元素乘法整合多个源的输入, 能表示任意排列, 同时参数量远小于全连接层。对于输入向量 x , 乘法层输出 $y \in \mathbb{R}^{S \times M}$:

$$y_{s,m} = \sum_i x_i D_{i,s} E_{i,m}, \text{ where } D \in \mathbb{R}^{d_{\text{model}} \times S}, E \in \mathbb{R}^{d_{\text{model}} \times M}$$

输出张量尺寸为 $\in \mathbb{R}^{\text{batch size} \times \text{length} \times S \times M}$, 随后接入一个二维卷积层 (length 和 S 分别对应图像高度与宽度), 进一步减少参数与计算量。下图展示了一个直观示意图: (a) 引入乘法层使每个分区可以访问任意嵌入维度; (b) 结合乘法层与 2D 卷积减少注意力层的参数量与计算时间。



此外, Scaling Transformer 还借鉴了 Reformer (Kitaev et al., 2020) 中的局部敏感哈希注意力 (LSH Attention) 以及循环结构的 FFN 块, 提出了新的架构 Terraformer, 以进一步支持长序列建模任务。

Mixture-of-Experts (MoE) 架构

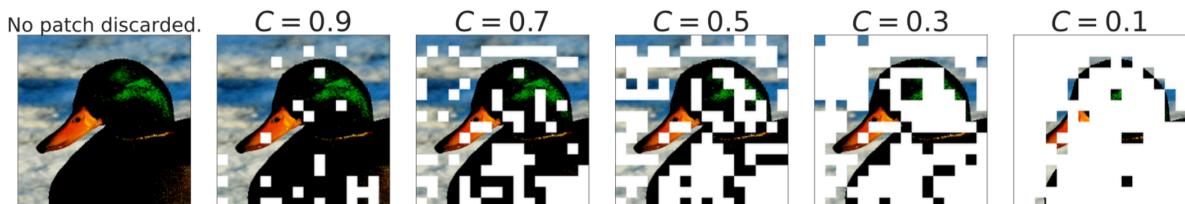
MoE 模型由多个专家子网络组成, 每个样本仅激活其中一部分进行推理, 从而在保持模型容量的同时降低计算成本。这一思想可追溯至 1991 年 (Jacobs et al.), 并在 Transformer 中被广泛采用 (详见 Fedus et al., 2022 综述)。每个专家的容量由容量超参数 capacity factor C 控制:

$$\text{Expert capacity} = \left\lceil C \cdot k \cdot \frac{\text{total } \# \text{ tokens in one batch}}{\#\text{experts}} \right\rceil$$

其中每个 token 会选择 top- k 个专家参与推理。若 $C > 1$, 表示专家容量冗余; 若 $C < 1$, 部分 token 可能因容量不足被舍弃。

路由策略改进 (Routing Strategy Improvement)

MoE 层使用路由网络将 token 分配给某些专家。在原始 MoE 中, token 按顺序依次分配。由于专家有容量限制, 如果重要 token 出现在较晚位置, 可能被丢弃。为解决此问题, **V-MoE (Vision MoE)** (Riquelme et al., 2021) 引入 **BPR (Batch Priority Routing)** 策略, 为每个 token 分配优先级分数 (如最大或 top- k 路由器得分), 再根据分数调整 token 顺序, 确保关键 token 优先占据专家容量。V-MoE 将 MoE 层应用到 Vision Transformer (ViT) 中, 性能达到 SoTA 但只需要的一半推理成本。模型最多扩展到 150 亿参数。它们在实验中设置 $k = 2$, 使用 32 个专家, 每隔一层插入一个 MoE 层。以下展示了一个图像 patches 根据优先级分数保留的例子。



BPR 在容量较低时 ($C \leq 0.5$) 效果远优于原始路由方案，能在较小容量下仍保持与稠密模型相当的性能。此外，观察发现，前面的 MoE 层倾向于提取通用特征，后面的 MoE 层则更可能对特定类别进行专业化。

Task MoE (Task-level Mixture-of-Experts) (Kudugunta et al. 2021) 在多语言神经机器翻译 (MNMT) 中，将传统的 Token MoE 路由策略扩展到任务层面 Task-level，将“任务”（例如目标语言）作为专家路由的基本单元，从而带来更高效的专家激活机制。

- **Token MoE** 为每个 token 独立选择专家，因而推理时需要加载所有专家参与计算；
- **Task MoE** 则基于任务统一决定专家分配，每个任务仅激活 Top- k 个专家，显著减少资源消耗。

实验表明 Task MoE 性能提升与 Token MoE 相当，且吞吐量提升 2.6 倍，解码器大小仅为原来的 1.6%。不过，该方法依赖预定义的任务划分（如语言对），对于没有明显任务边界的场景（如句子补全）不适用。

PR-MoE (Pyramid Residual MoE) (Rajbhandari et al. 2022) 提出了一种金字塔残差结构的 MoE 架构，旨在提升模型稳定性和训练效率。其关键思想是在每个 MoE 层中：

- 每个 token 始终经过一个**共享的标准 MLP**路径（即固定专家）；
- 同时选择一个动态路由的专家进行“加法残差增强”；
- 模型后层拥有更多的专家数量，实现**更高的计算能力与表达力**。

这样的设计既保留了标准网络的稳定性，又利用了 MoE 的参数可扩展性。如下为结构直观示意：



此外，PR-MoE 借助 DeepSpeed MoE 支持的**专家并行 + 数据并行**混合策略，允许不同层使用不同数量的专家资源，从而实现灵活的资源调度和更好的推理性能。

内核优化 (Kernel Optimization)

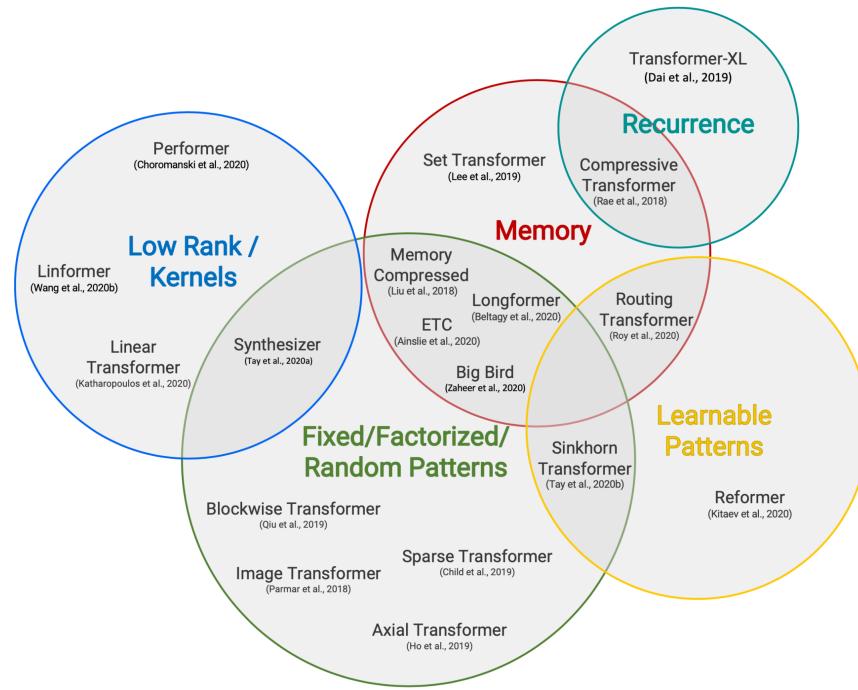
在大规模 Mixture-of-Experts (MoE) 模型中，**专家网络通常分布在多个 GPU 上**。随着 GPU 数量增加，每张卡上可容纳的专家数会减少，导致**跨设备通信变得更加频繁和昂贵**。尤其是在处理小批量数据时，通信延迟成为瓶颈，严重影响模型扩展效率。然而，现有通信库（如 NCCL 的 P2P API）在 NVLink 或 InfiniBand 等高带宽互连上，并不能充分发挥带宽优势，主要原因在于其未针对小工作负载和大规模集群优化。为此，研究者们提出了多种通信优化与内核加速技术，提高 MoE 的计算与通信效率：

DeepSpeed (Rajbhandari et al. 2022) 和 **TUTEL** (Hwang et al. 2022) 所采用的 **Hierarchical All-to-All** 通信方法，分为节点内和节点间两个阶段。这种分层设计将原本的通信跳数从 $O(G)$ 降为 $O(G_{node} + G/G_{node})$ ，其中 G 是 GPU 节点总数， G_{node} 是每个节点上的 GPU core 的数量。虽然该方法会引入额外通信量，但在实际训练中，由于瓶颈主要在于通信延迟而非带宽，因此该策略显著提高了小 batch 下的训练可扩展性。

DynaMoE (Kossmann et al. 2022) 通过 **动态重编译机制** (RECOMPILE) 优化专家之间的负载均衡问题。其核心思想包括：(1) **动态构建计算图**：在每轮训练前，根据样本被分配到哪些专家，动态构造计算图并重新分配资源，避免某些专家过载；(2) **动态调整容量因子 C**：在训练中，DynaMoE 根据专家负载自动调整每个专家可接收 token 的数量，减少计算冗余和显存消耗；(3) **缓存样本分配**：实验发现，大多数样本在训练初期就稳定地分配给特定专家，因此可以缓存这些分配结果，加速后续阶段的路由过程。(4) **移除 gating 网络**：在专家分配足够稳定之后，可使用 RECOMPILE 去除 gating 网络，从而进一步提升推理效率并简化模型图。

架构优化 (Architectural Optimization)

随着模型规模不断增长，Transformer 的计算与内存瓶颈问题日益凸显。为提升模型推理效率与可扩展性，研究者们提出了大量 **架构优化 (Architectural Optimization)** 方法。Tay et al. 2020 在其综述论文《Efficient Transformers》中系统梳理了这类优化策略，成为该领域的重要参考文献。下图总结了他们提出的高效 Transformer 分类方法：



稀疏注意力模式 (Sparse Attention Patterns)

稀疏注意力通过限制每个 token 仅与部分 token 建立注意力，以此缓解注意力矩阵的二次复杂度。可分为三类：

- **固定模式 (Fixed Patterns)** 使用预定义的稀疏连接结构，不随输入变化，适配性强，计算友好
 - **Blockwise Attention**：输入序列被划分为固定大小的块，每个块内部全连接（局部注意力）；
 - **Image Transformer**：针对图像 patch 的局部窗口注意力；
 - **Sparse Transformer**：使用跨步（strided）注意力模式。
- **组合模式 (Combined Patterns)** 将多个固定模式组合，以提高注意力的表达范围和灵活性：
 - **Sparse Transformer**：结合了跨步和局部注意力；
 - **Axial Transformer**：对高维张量在每个维度方向分别计算注意力（轴向注意力）；

- ETC / Longformer / Big Bird：将局部注意力、全局 token 和稀疏连接结合，达到兼顾效率与性能的目的；
- 可学习模式（Learnable Patterns）注意力连接不再预设，而是由模型动态学习获得：
 - Reformer：使用局部敏感哈希（LSH）将相似 token 聚类，仅在组内计算注意力；
 - Routing Transformer：通过 K-Means 聚类动态组织注意力；
 - Sinkhorn Transformer：基于 Sinkhorn 排序网络（Sorting Network）对 token 排序并分块，以学习连接结构。

循环机制（Recurrence）

通过引入循环（如 RNN-style 记忆）延续跨块上下文，从而突破原始 Transformer 的固定上下文窗口限制：

- Transformer-XL 引入段间状态缓存，允许模型跨 segment 存储并访问历史激活；
- Universal Transformer 将自注意力与 RNN 中的循环机制相结合；
- Compressive Transformer 在 Transformer-XL 基础上引入压缩记忆（compressed memory），长期历史被压缩保存；

内存优化结构（Memory Saving Designs）

通过结构设计来减少内存占用：

- Linformer：对 K, V 进行低秩投影 ($n \times d \rightarrow k \times d$)，使注意力复杂度从 $O(n^2)$ 降为 $O(nk)$ ；
- Multi-Query Attention：多个注意力头共享同一组 Key 和 Value，可显著减少张量大小；
- Performer：使用核函数（Kernel methods）将标准注意力转为线性形式（线性注意力），实现近似计算但大幅节省内存。

自适应注意力（Adaptive Attention）

自适应注意力机制允许模型根据输入 token 动态调整注意力范围或计算深度，实现更加灵活的计算资源分配。

- Adaptive Attention Span：每个注意力头拥有可学习的“注意力窗口长度”，训练时可被自动裁剪；
- Universal Transformer 融合循环机制并引入 ACT (Adaptive computation time) 动态决定循环次数；
- Depth-Adaptive Transformer and CALM 根据置信度评估决定是否对每个 token 提前退出计算层，从而实现动态推理。

References

- [1] Bondarenko et al. "Understanding and overcoming the challenges of efficient transformer quantization" ACL 2021.
- [2] Dettmers et al. "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale" NeurIPS 2022
- [3] Zadeh et al. "Gobo: Quantizing attention-based NLP models for low latency and energy efficient inference." MICRO 2020
- [4] Shen, Dong & Ye, et al. "Q-BERT: Hessian based ultra low precision quantization of BERT" AAAI 2020.
- [5] Yao et al. "ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers" arXiv preprint arXiv:2206.01861 (2022).

- [6] Frantar et al. "GPTQ: Accurate Quantization for Generative Pre-trained Transformers" arXiv preprint arXiv:2210.17323 (2022).
- [7] Xiao & Lin "SmoothQuant: Accelerated sparse neural training: A provable and efficient method to find N:M transposable masks." arXiv preprint arXiv:2211.10438 (2022). | [code](#)
- [8] Pool & Yu. "Channel Permutations for N:M Sparsity." NeurIPS 2021. | [code](#)
- [9] Zhou & Ma, et al. "Learning N:M fine-grained structured sparse neural networks from scratch." arXiv preprint arXiv:2102.04010 (2021).
- [10] Jayakumar et al. "Top-KAST: Top-K Always Sparse Training." NeurIPS 2020.
- [11] Nvidia. "Nvidia A100 tensor core GPU architecture." 2020.
- [12] Gale, Elsen & Hooker "The State of Sparsity in Deep Neural Networks." arXiv preprint arXiv:1902.09574 (2019).
- [13] Zhu & Gupta. "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression." arXiv preprint arXiv:1710.01878 (2017).
- [14] Renda et al. "Comparing rewinding and fine-tuning in neural network pruning." arXiv preprint arXiv:2003.02389 (2020).
- [15] Zhou & Ma, et al. "Learning N:M fine-grained structured sparse neural networks from scratch." arXiv preprint arXiv:2102.04010 (2021).
- [16] Pool & Yu. "Channel Permutations for N:M Sparsity." NeurIPS 2021. | [code](#)
- [17] Jaszczur et al. "Sparse is Enough in Scaling Transformers." NeurIPS 2021.
- [18] Mishra et al. "An Survey of Neural Network Compression." arXiv preprint arXiv:1710.09282 (2017).
- [19] Fedus et al. "A Review of Sparse Expert Models in Deep Learning." arXiv preprint arXiv:2209.01667 (2022)..
- [20] Riquelme et al. "Scaling vision with sparse mixture of experts." NeurIPS 2021.
- [21] Kudugunta et al. "Beyond Distillation: Task-level Mixture-of-Experts for Efficient Inference." arXiv preprint arXiv:2110.03742 (2021).
- [22] Rajbhandari et al. "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale." arXiv preprint arXiv:2201.05596 (2022).
- [23] Kossmann et al. "Optimizing mixture of experts using dynamic recompilations." arXiv preprint arXiv:2205.01848 (2022).
- [24] Hwang et al. "Tutel: Adaptive mixture-of-experts at scale." arXiv preprint arXiv:2206.03382 (2022). | [code](#)
- [25] Noam Shazeer. "Fast Transformer Decoding: One Write-Head is All You Need." arXiv preprint arXiv:1911.02150 (2019).
- [26] Tay et al. "Efficient Transformers: A Survey." ACM Computing Surveys 55.6 (2022): 1-28.
- [27] Pope et al. "Efficiently Scaling Transformer Inference." arXiv preprint arXiv:2211.05102 (2022).
- [28] Frankle & Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks" ICLR 2019.
- [29] Elabyad et al. "Depth-Adaptive Transformer" ICLR 2020.
- [30] Schuster et al. "Confident Adaptive Language Modeling" arXiv preprint arXiv:2207.07061 (2022).

- [31] Gou et al. "<https://arxiv.org/abs/2006.05525>" arXiv preprint arXiv:2006.05525 (2020).
- [32] Hinton et al. "Distilling the Knowledge in a Neural Network" NIPS 2014.
- [33] Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter" Workshop on Energy Efficient Machine Learning and Cognitive Computing @ NeurIPS 2019.