机器学习系统 2026春

第五章 混合专家模型

张燕咏 讲席教授 yyz@ustc.edu.cn
张午阳 特任教授 wuyangz@ustc.edu.cn

中国科学技术大学
University of Science and Technology of China

PART 01

为什么需要混合专家模型?

从Scaling Laws到条件计算 -- 理解MoE的核心动机

Scaling Laws: Dense模型的根本困境

背景: 2020年Kaplan等人(OpenAI)做了一个系统性实验: 训练数百个不同规模的语言模型, 发现性能与三个因素之间存在幂律关系(Power Laws)。这些定律揭示了Dense模型的根本困境 — 提升性能必须等比例增加计算量。

三大Scaling Laws

1. 参数量定律 (Parameters)

$L(N) = (N_c / N)^{\alpha_N}, \alpha_N \approx 0.076$

含义: 模型参数翻倍, 损失仅下降约5%。需要指数级增长参数才能获得线性改进。

2. 数据量定律 (Dataset Size)

$L(D) = (D_c / D)^{\alpha_D}, \alpha_D \approx 0.095$

含义: 数据量翻倍, 损失下降约6.5%。互联网文本终究有限, 数据墙(Data Wall)日益逼近。

3. 计算量定律 (Compute Budget)

$L(C) = (C_c / C)^{\alpha_C}, \alpha_C \approx 0.050$

含义: 计算翻倍, 损失仅下降约3.4%。这是Dense模型最昂贵的维度。

Chinchilla修正 (2022, Hoffmann)

Kaplan认为优先扩参数; Chinchilla发现数据同样重要:
$N_{opt} \propto C^{0.50}, D_{opt} \propto C^{0.50}$

即: 参数和数据应等比例增长。Chinchilla (70B, 1.4T tokens) 击败了Gopher (280B, 300B tokens)。

但这依然无法逃脱: 2× 性能 → 4× 成本。

Dense模型的根本瓶颈

Dense模型的每次前向传播都激活全部参数:
$C_{forward} \approx 2N$ (FLOPs per token)

问题: 想让模型"更聪明"(更多参数N) → 必然"更慢"(更多计算C)
参数量与计算量完全耦合 — 这是Dense架构的先天缺陷。
核心问题: 能否让参数增长, 但计算量基本不变?

关键洞察: Scaling Laws表明Dense模型面临指数级成本增长。MoE的核心目标 = 将参数量与计算量解耦 (decouple parameters from compute)。

Sources: Kaplan et al. 2020; Hoffmann et al. 2022 (Chinchilla)

MoE的解决方案: 条件计算 (Conditional Computation)

背景: 2013年Bengio等人首次提出"条件计算"(Conditional Computation)的理念: 不必对每个输入都使用全部模型参数, 而是根据输入内容动态选择一部分子网络来处理。MoE正是这一理念最成功的实现。

Dense模型 (传统方式)

- **每个token都经过全部参数计算**
  激活参数 = 总参数 → 无法分离
  典型: GPT-3 175B → 每token 350B FLOPs
  LLaMA-2 70B → 每token 140B FLOPs
  扩展方式: 增加层数或隐藏维度
  → 训练时间、推理延迟线性增长
  → GPU显存需求线性增长
  → 性能提升边际递减(Scaling Laws)

MoE模型 (稀疏激活)

- **每个token只激活一小部分专家**
  激活参数 << 总参数 → 成功解耦!
  Mixtral 8×7B: 47B总参数, 仅激活13B
  DeepSeek-V3: 671B总参数, 仅激活37B
  扩展方式: 增加专家数量 (Expert Count)
  → 计算量几乎不变 (每token FLOPs固定)
  → 模型"知识容量"随专家数增长
  → 训练效率提升4-7倍 (同等质量)

模型	总参数	激活参数	激活比	专家数	Top-K	相当Dense
Switch-C	1.57T	~1.6B	0.1%	2048	1	~1.6B
Mixtral 8×7B	46.7B	12.9B	27.6%	8	2	~13B
DeepSeek-V2	236B	21B	8.9%	160	6	~21B
DeepSeek-V3	671B	37B	5.5%	256	8	~37B
DBRX	132B	36B	27.3%	16	4	~36B
Grok-1	314B	~86B	27.4%	8	2	~86B

MoE核心优势: 用1/5~1/10的计算成本, 达到Dense模型同等甚至更好的性能。参数=知识容量, 计算=推理速度, 二者终于解耦。

数据来源: 各模型官方论文与技术报告, 2023-2024

MoE核心公式: 一个公式理解混合专家

核心思想: MoE的数学原理可以浓缩为一个公式。Router根据输入x计算每个专家的权重G(x), 然后将Top-K个专家的输出加权求和。关键在于: 无论有多少个专家, 每个token只激活K个, 计算量与K成正比, 与总专家数N无关。

MoE输出公式

$$y = \Sigma_i\ G(x)_i \cdot E_i(x) \quad \text{(仅对 Top-K 个 i 求和)}$$

符号说明:

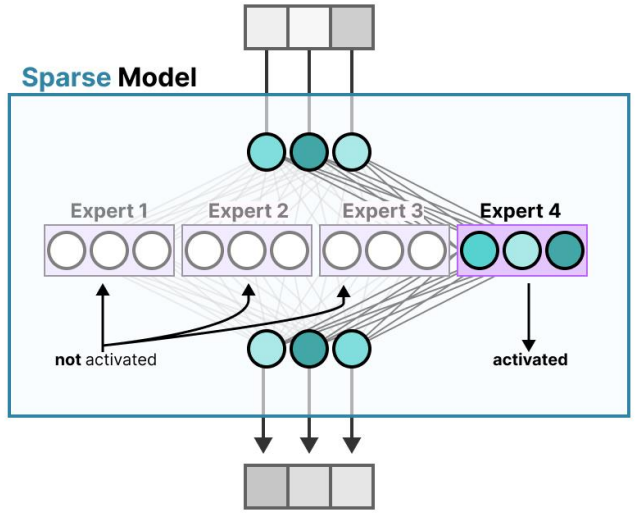
x – 输入token的隐藏状态向量 (hidden state)
$E_i(x)$ – 第i个专家网络的输出 (通常是FFN)
$G(x)_i$ – 门控/路由网络为专家i分配的权重
N – 专家总数 (8, 16, 64, 2048 ...)
K – 每个token激活的专家数

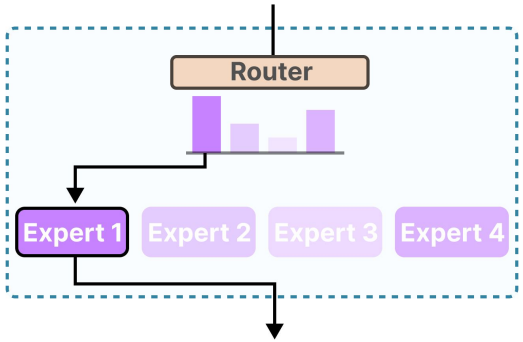
为什么计算不随N增长?

G(x)对非Top-K的专家输出 = 0
→ 只需计算K个专家, 其余专家完全跳过
→ FLOPs ∝ K × d_expert, 与N无关!

路由机制演进:

Shazeer 2017: K=4 (激活4/2048个专家)
GShard 2020: K=2 (Top-2路由成为主流)
Switch 2021: K=1 (极致稀疏, 最高效率)
趋势: K越小 → 路由越简单 → 负载均衡越关键



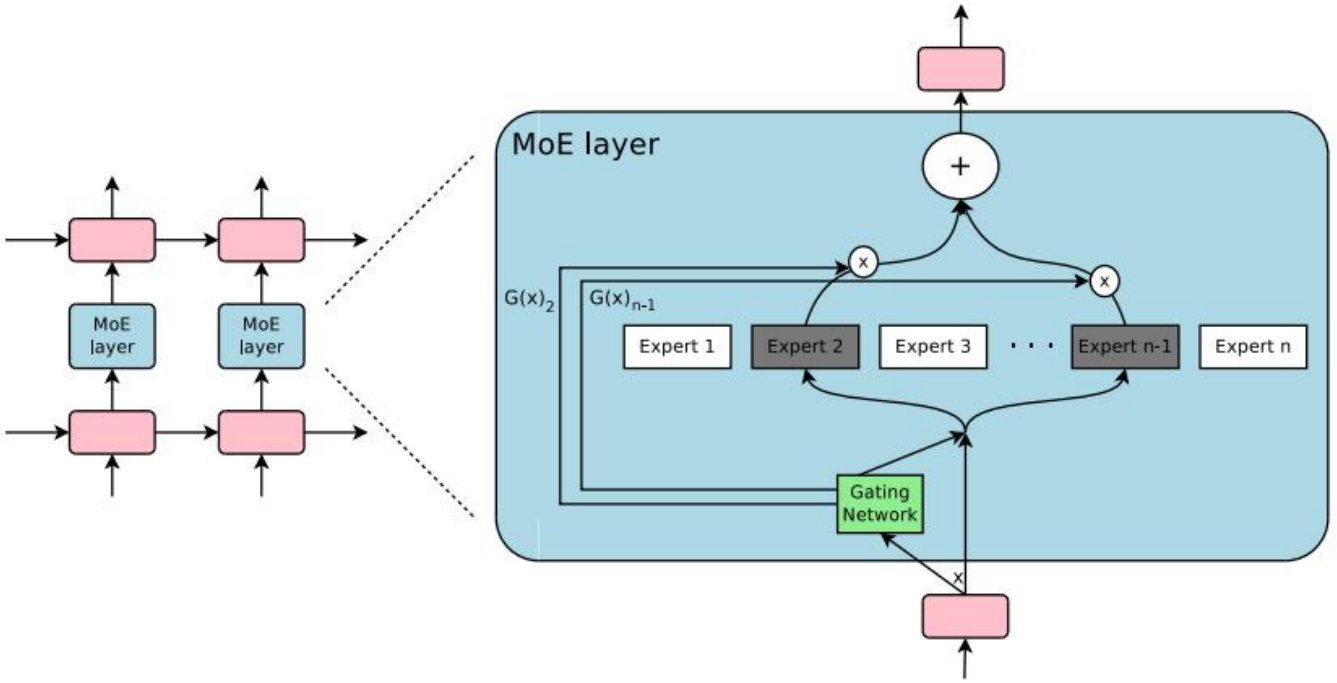


Images: Maarten Grootendorst, A Visual Guide to MoE, 2024

PART 02

MoE发展历程

1991→2025: 从局部专家到万亿参数



MoE发展里程碑: 35年演进

1991 — 奠基

Jacobs et al.: Mixture of Experts

首次提出MoE概念: 多个专家网络 + 竞争性门控, 用于回归/分类任务。奠定了"分而治之"的基础思想。

1994 — 理论

Jordan & Jacobs: Hierarchical MoE

引入层次结构的MoE, 使用EM算法训练。专家可以递归地包含子专家, 形成树状结构。

2017 — 突破

Shazeer et al.: Outrageously Large NNs

首个大规模稀疏MoE: 2048专家, Top-K门控, 负载均衡损失。在翻译和语言建模任务中证明了MoE的有效性。

2020 — 系统

Lepikhin et al.: GShard

首个MoE系统论文: Expert Parallelism, All-to-All通信, 600B参数模型在2048 TPU上4天训练完成。

2021 — 里程碑

Fedus et al.: Switch Transformer

极简路由: Top-1 (每token仅1个专家), 1.57T参数, 训练速度比T5-XXL快7倍, 首次突破万亿参数。

2022 — 优化

Zoph / Zhou: ST-MoE / Expert Choice

ST-MoE: 详尽的MoE训练稳定性研究。Expert Choice: 反转路由 — 让专家选token而非token选专家。

2024 — 产业化

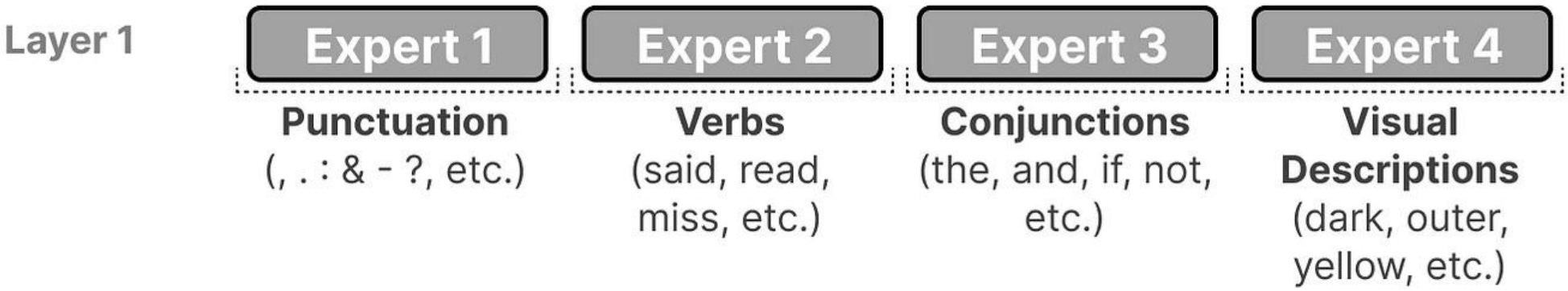
DeepSeek / Mistral: DeepSeek-V2/V3, Mixtral

DeepSeek-V3: 671B参数, Fine-grained Experts(256个) + 辅助无损负载均衡。Mixtral: 开源标杆, 8×7B。

PART 03

MoE核心组件

专家网络 · 门控机制 · Transformer集成



专家网络与Transformer集成

核心思想: 在标准Transformer中, MoE层替换掉每个Block中的FFN (前馈网络)。每个"专家"本质上就是一个独立的FFN, 它们共享相同的输入/输出维度, 但拥有各自独立的权重。通过路由机制, 每个token只激活少量专家。

Expert = 独立的FFN

标准FFN结构:

$FFN(x) = W_2 \cdot \sigma(W_1 x + b_1) + b_2$

其中 σ 为激活函数 (ReLU/GELU/SwiGLU)

MoE中每个专家E_i都是完整FFN:

拥有独立的 $W_1^{(i)}, W_2^{(i)}, b_1^{(i)}, b_2^{(i)}$

共享输入/输出维度 d_model

专家间不共享权重 → 各自学习不同知识

放置策略 (Placement Strategy):

每隔一层替换 (Every Other Layer)

Mixtral / GShard采用, MoE层与Dense FFN交替

优点: 稳定性好, Dense层提供"共享知识"

所有层替换 (All Layers)

Switch Transformer采用, 全部FFN→MoE

优点: 最大稀疏化, 参数效率最高

仅后半部分 (Back Half Only)

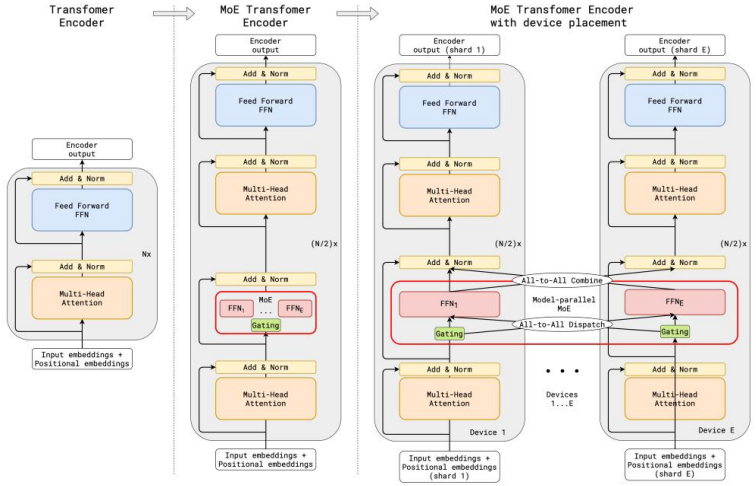
某些实验表明后层专家分工更明显

Switch计算效率示例:

128专家 × 58M/专家 = 7,410M 总参数

但每token仅激活 1 专家 = 223M FLOPs

→ 33× 参数量, 同等计算成本!



Expert specialization	Expert position	Routed tokens
Punctuation	Layer 2	, , , , , , , , ,- , , , , ,).)
	Layer 6	, , , , , : . : , & , & & ? &-,,? ,,, .
Conjunctions and articles	Layer 3	The the the the the the the the the The...
	Layer 6	a and and and and and and or and ...
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died...
Visual descriptions *color, spatial position*	Layer 0	her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue
Counting and numbers *written and numerical forms*	Layer 1	after 37 19. 6. 27 I I Seven 25 4, 54 I two dead we Some 2012 who we few lower

Images: HuggingFace MoE Blog; Maarten Grootendorst

门控网络 / 路由机制 (Gating Network / Router)

关键挑战: Router是MoE中最关键也最脆弱的组件。它决定每个token交给哪些专家处理。一个差的路由策略会导致"专家崩塌" — 所有token被路由到同一个专家，其余专家从未被训练，白白浪费参数。

从Softmax到Noisy Top-K路由

Step 1: 计算原始得分 (Logits)

$h(x) = W_g \cdot x \quad (W_g \in \mathbb{R}^{N \times d})$

Router就是一个简单的线性层!

Step 2: 加噪声 + Top-K (Noisy Top-K)

$H(x)_i = h(x)_i + \varepsilon \cdot \text{Softplus}(W_{noise} \cdot x)_i$

其中 $\varepsilon \sim N(0,1)$, $\text{Softplus}(z) = \ln(1+e^z)$

噪声的作用:

- 训练早期: 鼓励token探索不同专家
- 防止"赢者通吃": 避免强专家越来越强
- 噪声量可学习: 模型自动调节探索程度

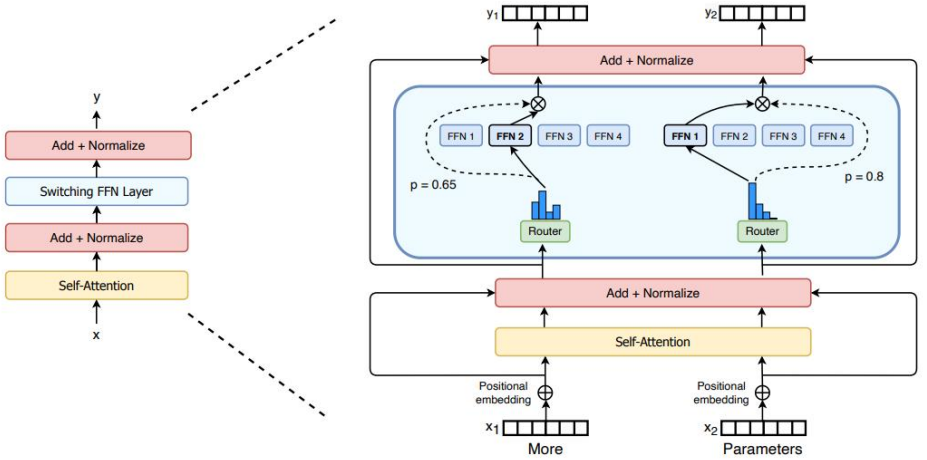
Step 3: 归一化Top-K概率

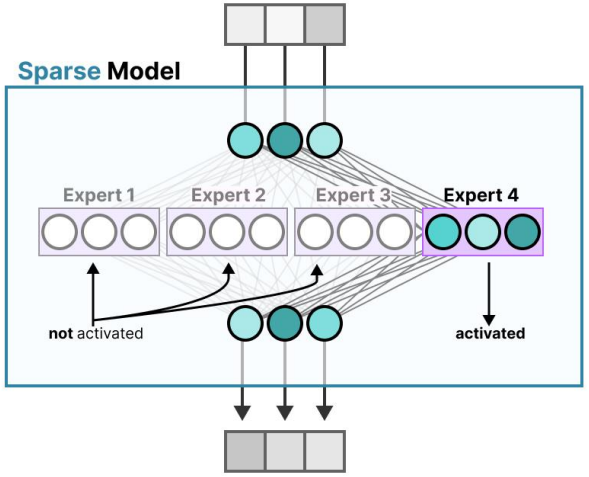
$G(x)_i = \text{Softmax}(\text{TopK}(H(x), K))_i$

非Top-K的$G(x)_i = 0 \rightarrow$ 完全不计算

路由简化趋势:

Shazeer 2017: Noisy Top-K + 辅助损失
Switch 2021: Top-1 + 简化辅助损失
Expert Choice:反转路由，专家选token
DeepSeek: 无辅助损失的负载均衡



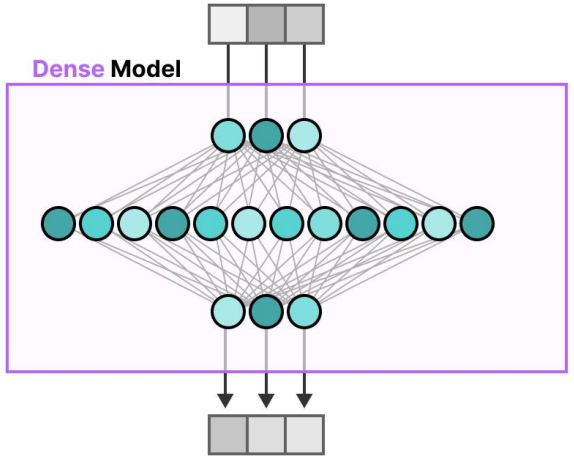


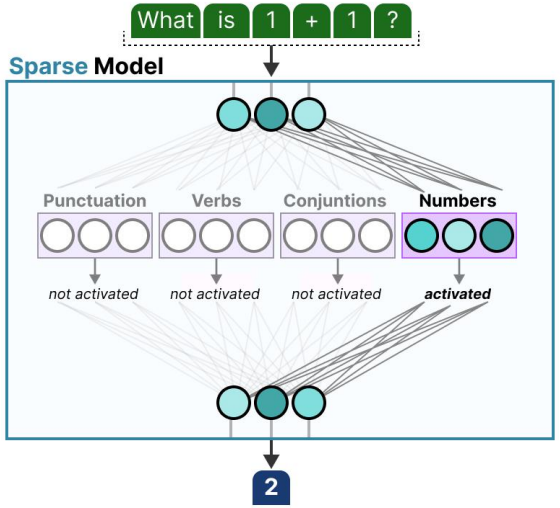
Images: HuggingFace Switch Transformer; Maarten Grootendorst

PART 04

Shazeer 2017: 稀疏门控MoE

首个大规模MoE的突破性工作





Shazeer 2017: 负载均衡 — MoE训练的核心挑战

为什么需要负载均衡? 没有负载均衡时, MoE训练会陷入"自我强化崩塌"(Self-reinforcing Collapse): 某个专家因为初始化偶然表现稍好 → 更多token被路由到它 → 它得到更多训练 → 表现更好 → 吸引更多token... 最终99%的token流向1-2个专家, 其余专家"饿死"。

两个辅助损失函数

1. Importance Loss — 确保门控权重均衡

$$Importance(X, i) = \Sigma_x G(x)_i$$

$$L\_imp = w\_imp \cdot CV(Importance(X))^2$$

CV = 变异系数 (标准差/均值)

目标: 让每个专家获得的门控权重之和相近

2. Load Loss — 确保实际token数均衡

$$Load(X, i) = \Sigma_x P(x, i)$$

$P(x,i) = Pr(专家i在token\ x的Top\text{-}K中)$

$$L\_load = w\_load \cdot CV(Load(X))^2$$

目标: 让每个专家处理的token数相近

实验结果 (Table 6, 1B Word):

配置	w_imp	w_load	Perplexity	Max/Mean Load
无负载均衡	0	0	39.8	17.80
仅Importance	0.1	0	36.4	2.05
Imp + Load	0.1	0.1	35.6	1.14

自我强化崩塌循环

崩塌机制 (Self-reinforcing Collapse):

初始化 → 专家E_3偶然表现稍好

↓

Router给E_3更高权重

↓

更多token路由到E_3

↓

E_3获得更多训练梯度

↓

E_3表现进一步提升

↓

其他专家"饿死", 从不被训练

↓

最终: 1-2个专家处理99%的token
MoE退化为一个小型Dense模型!

解决方案:
辅助损失迫使Router均匀分配
噪声注入增加探索性
容量因子限制每专家最大token数

核心发现: 同时使用Importance Loss + Load Loss, perplexity从39.8降至35.6, 负载比从17.8:1降至1.14:1。

Shazeer 2017: 关键实验结果

问题: 首个大规模MoE到底取得了什么成果? Shazeer在语言建模和机器翻译两大任务上进行了系统实验, 证明MoE可以在相同计算预算下显著超越Dense模型。

任务 / 模型	总参数	每步运算量	专家数	指标	结果
LM - Flat MoE	4.37B	142.7M	256	Perplexity	**28.0**
LM - Dense Baseline	~151M	142.7M	-	Perplexity	30.6
LM - Hier. MoE	4.37B	142.7M	256+16	Perplexity	**27.6**
翻译 - MoE En→Fr	8.7B	~213M	2048	BLEU	**40.56**
翻译 - GNMT Baseline	~278M	~213M	-	BLEU	39.22

核心发现

语言建模 (1B Word Benchmark):
MoE Perplexity 28.0 vs Dense 30.6 (↓8.5%)
同等计算量, MoE参数量为Dense的29倍
层级MoE更优: Perplexity 27.6

机器翻译 (WMT En→Fr):
BLEU 40.56 vs GNMT 39.22 (↑1.34)
在当时创下SOTA记录
使用2048个专家, 总参数8.7B

大规模验证 (100B语料):
使用65536个专家: Perplexity降低39%
证明MoE可以持续通过增加专家数扩展

专家专业化现象 (Expert Specialization)

专家自动学会"分工":
语法专家: 处理功能词
实体专家: 处理专有名词
数字专家: 处理数值token
句尾专家: 处理标点边界

这种专业化是自发涌现的!
Router通过梯度下降自动学会分工策略。

意义: MoE不仅提高效率, 还能学到更好的表征。

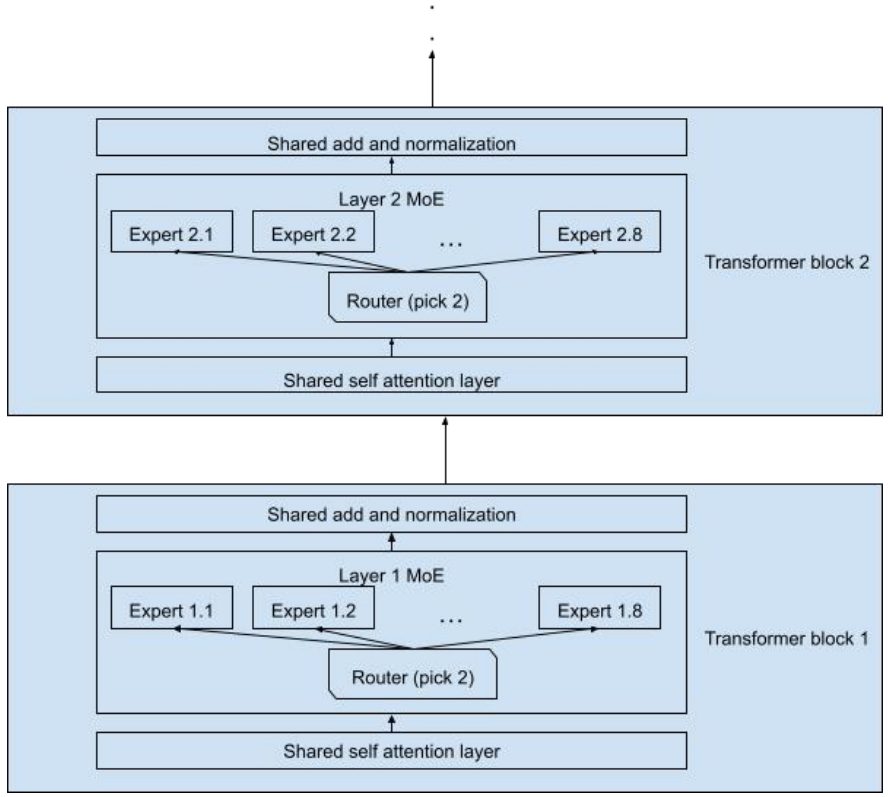
Shazeer 2017的核心贡献: 证明MoE在大规模NLP任务中可行, 引入Noisy Top-K路由+辅助损失, 奠定了现代MoE的技术基础。

Source: Shazeer et al., Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, ICLR 2017

PART 05

GShard: 首个MoE系统设计

Expert Parallelism与All-to-All通信



GShard: Expert Parallelism与系统设计

核心定位: GShard (Lepikhin et al., 2020) 本质上是一篇系统设计论文。它首次解决了"如何在数千台设备上高效训练MoE"的工程问题，提出了Expert Parallelism — 将不同专家放置在不同设备上，通过All-to-All通信交换token。

Expert Parallelism 核心概念

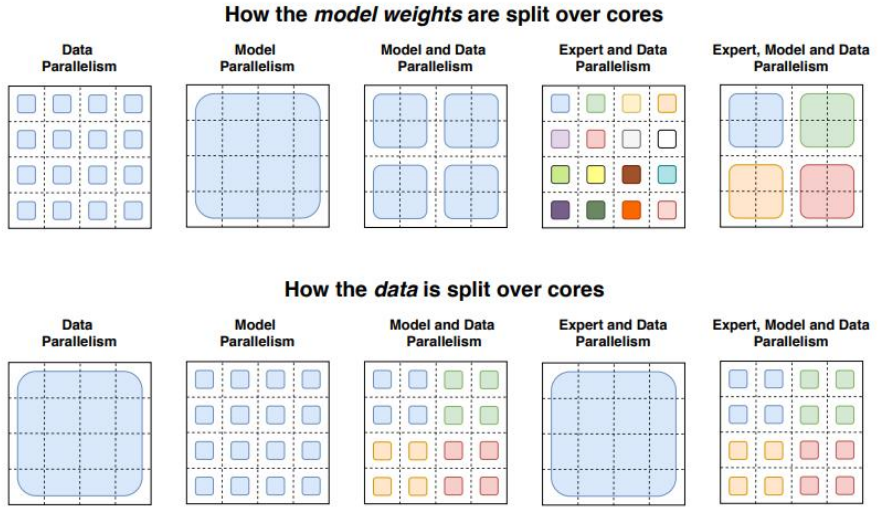
Expert Parallelism (EP):
将N个专家分布到N个设备上
每个设备负责1个(或几个)专家的计算
非MoE层 (Attention等) 仍用Data Parallelism

All-to-All 通信模式:
Dispatch: 每个设备将token发送给对应专家所在设备
Compute: 各设备上的专家独立计算
Combine: 专家输出发回原始设备，加权求和

关键设计: 容量因子 (Capacity Factor)
每个专家的缓冲区大小 = CF × (T/N)

T=总token数，N=专家数，CF通常=1.0~2.0
溢出token被丢弃，走残差连接 (skip)
CF↑: 减少token丢弃但浪费计算/内存

随机路由 (Random Routing):
Top-2中: 第1专家必定选择
第2专家以其门控概率随机决定是否选择
推理时节省50%计算，质量损失极小



指标	GShard MoE	Dense对比
模型规模	**600B参数**	~6B参数
设备数	2048 TPUv3	2048 TPUv3
训练时间	4天	42天 (估)
训练成本	22.4 TPU-yrs	235.5 TPU-yrs
翻译质量 (BLEU)	**+13.5 (100语对)**	Baseline
MoE层比例	每隔一层	全部Dense
专家路由	Top-2 + Random	-

GShard核心贡献: Expert Parallelism使MoE可以扩展到数千设备，600B模型仅需4天 (vs Dense 42天)，训练成本降低10.5倍。

Source: Lepikhin et al., GShard: Scaling Giant Models with Conditional Computation, ICLR 2021

1.5 GShard: Top-2门控与大规模翻译结果

背景: GShard (Lepikhin 2020) 是首个将MoE成功应用于Transformer的大规模系统。其核心创新是Top-2门控: 每个token选择2个专家, 第一个由置信度决定, 第二个随机选择(概率正比于门控值)。这一设计在保证性能的同时大幅简化了通信模式。

Top-2门控算法 (Algorithm 1)

对每个token x:

- 1. 计算门控值: g = Softmax(x * W_g)
- 2. 第一专家: e1 = argmax(g), 权重 w1 = g[e1]
- 3. 第二专家: 随机采样, P(e2=i) 正比于 g[i]
- 4. 权重归一化: w2 = g[e2] / (w1 + g[e2])

容量因子 (Capacity Factor, CF)

每个专家的缓冲区大小:

capacity = CF * (tokens_in_batch / num_experts)

- CF=2.0 (GShard默认), 超出的token被丢弃

辅助负载均衡损失 l_aux

l_aux = (1/E) * Sum_e (c_e/S) * m_e

- c_e: 分配给专家e的token数
- S: batch中总token数
- m_e: 门控概率均值, 促使路由更均衡

GShard大规模实验结果

模型	参数量	BLEU	训练时间
Dense Baseline	2.3B	36.9	42天
GShard Top-2	**600B**	**44.3**	**4天**

结果分析:

- BLEU提升: +7.4 (相对提升+20%)
- **训练加速: 42天 -> 4天 (10.5x)**
- 100语言翻译均值BLEU 44.3
- 模型规模: 260x参数, 近似计算量

硬件配置:

- 2048 TPU v3, 每TPU ~1专家, SPMD范式

关键要点: GShard证明系统设计与算法同等重要: Top-2路由 + SPMD编译 + 2048 TPU, 42天→4天, BLEU+20%。

1.5 GShard系统设计: All-to-All通信与SPMD编译

系统视角: GShard的核心贡献不仅是MoE算法, 更是将MoE与分布式系统深度结合的工程范式。它解决了两个关键问题: (1) 如何在数千设备间高效调度token? (2) 如何让编译时间不随设备数增长? 这两个问题的解决方案影响了后续所有大规模MoE系统。

All-to-All通信模式 (三步流程)

Step 1 - Dispatch (分发):
每个设备根据路由结果, 将token发送到对应专家所在设备
通信模式: All-to-All (每对设备间都有数据交换)

Step 2 - Expert Compute (计算):
每个专家处理分配到的token (本地计算)
各专家完全并行, 无依赖

Step 3 - Combine (聚合):
专家输出通过反向All-to-All返回原设备
加权求和得到最终输出

SPMD编译 (关键系统创新)

SPMD = Single Program Multiple Data

- 同一程序在所有设备上运行, 仅数据不同
- **编译时间O(1) -- 与设备数无关!**
- vs MPMD: 编译时间O(D), D为设备数

通信开销分析

All-to-All通信量: O(B * d_model)

- 每token发送: 2 * d_model floats (Top-2路由)
- 随设备数增长: O(sqrt(D)) 带宽需求

扩展性数据:

设备数	专家数	每设备内存	步进时间
128	128	~恒定	1.0x (基准)
512	512	~恒定	~1.3x
2048	2048	~恒定	1.7x

关键发现:

- **128->2048设备 (16x), 步进时间仅1.7x**
- 每设备内存恒定: 专家分布式存储
- 近似线性扩展的计算吞吐量

关键要点: GShard的SPMD编译 + All-to-All通信模式成为MoE分布式系统的标准范式。16x设备扩展仅1.7x步进时间开销。

Lepikhin et al. (2020) arXiv:2006.16668 | Figure adapted from HuggingFace Blog

PART 06

Switch Transformer: 简化与规模化

Top-1路由 * 容量因子 * 万亿参数

1.6 Switch Transformer: Top-1路由 -- 更简单即更好

背景: Shazeer (2017) 曾推测K>1(即同时选择多个专家)是MoE成功的必要条件。Switch Transformer (Fedus 2021) 大胆挑战了这一假设，证明仅选择1个专家(Top-1)不仅足够，而且在速度、通信和扩展性上全面优于Top-2路由。

Top-1路由的三大优势

1. 计算量减半

- Top-2: 每token计算2个FFN = 2x FLOPs
- Top-1: 每token计算1个FFN = 1x FLOPs
- **同等FLOPs下可容纳2x更多专家**

2. 通信量减半

- 每个token只需发送给1个设备 (而非2个)
- All-to-All数据量直接减半
- 多节点场景下优势更为显著

3. 路由逻辑更简单

- 无需第二专家的随机采样
- 无需多专家权重归一化
- 实现更容易、调试更方便

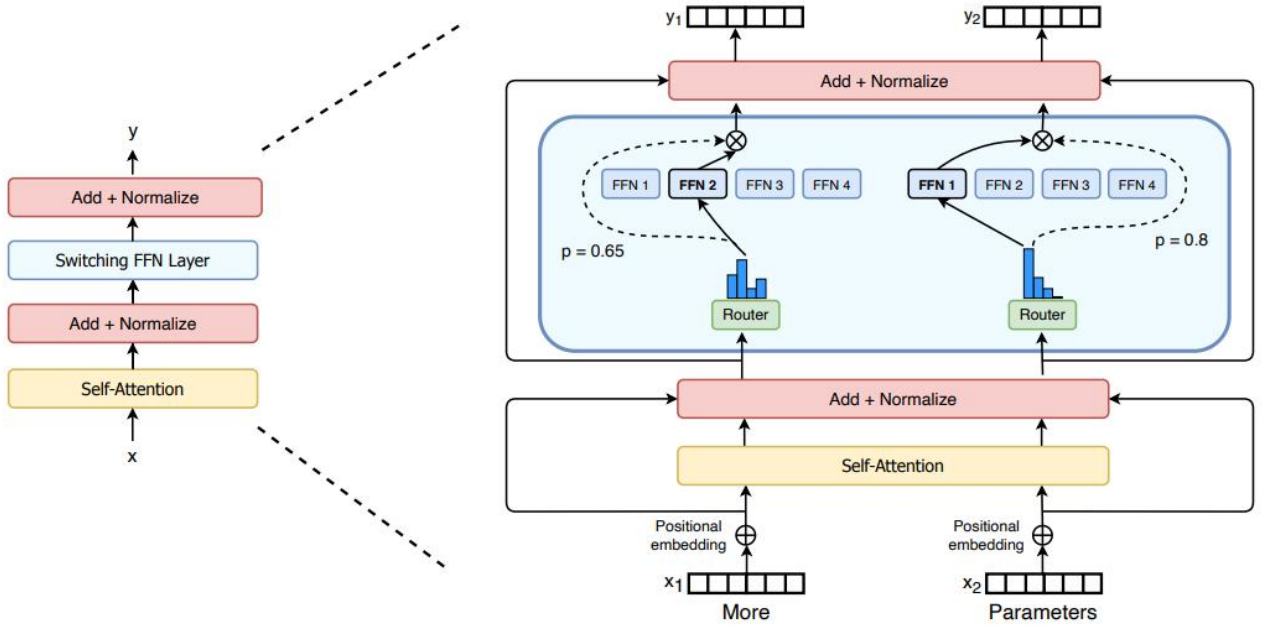
Switch路由公式:

p = Softmax(x * W_r)

expert = argmax(p), weight = p[expert]

output = weight * FFN_expert(x)

实测: 7x预训练加速，相同FLOPs!



关键要点: Switch Transformer证明K=1足够: 计算和通信都减半，预训练加速7x。简单性是大规模系统的关键设计原则。

Fedus et al. (2021) Switch Transformers arXiv:2101.03961 | 图: HuggingFace Blog

1.6 容量因子与Token Dropping: 处理路由不均衡

背景: 当路由不均匀时, 某些专家会收到过多token, 超出其处理能力。Switch Transformer引入容量因子(Capacity Factor, CF)来控制每个专家的缓冲区大小。超出容量的token被丢弃(通过残差连接直接跳过MoE层), 实验表明模型对10-15%的token丢弃非常鲁棒。

容量公式 (Expert Buffer Size)

expert_capacity = CF * (tokens / experts)

例: 512 tokens, 128 experts, CF=1.25

- 理想均匀分配: 512/128 = 4 tokens/expert
- 实际容量: 1.25 * 4 = 5 tokens/expert
- 超过5个token的部分被丢弃

Token Dropping机制:

被丢弃的token不经过专家, 而是:
output = residual(x) (直接跳过MoE层)

- token的信息通过残差连接保留
- 后续层仍可正常处理该token
- **实验: 丢弃10-15% token质量几乎不变!**

负载均衡损失 (Switch版本):

L_balance = alpha * N * Sum(f_i * P_i)

- f_i: 实际分配到专家i的token比例
- P_i: 路由到专家i的平均概率
- alpha = 0.01 (默认), 过大会损害质量

容量因子 (CF) 取值对比

CF值	缓冲大小	内存开销	丢弃率	质量
1.0	最小	最低	较高 ~20%	稍低
1.25	适中	适中	+25%余量	**最优**
1.5	较大	较高	+50%余量	略好
2.0	最大	最高	几乎无丢弃	过度开销

CF = 1.25 是最佳平衡点:

- 内存开销仅增25%, 丢弃率大幅下降
- 质量优于CF=1.0, 接近CF=1.5
- 实际训练中token丢弃率<5%

为什么丢弃是可接受的?

- 残差连接保证信息不完全丢失
- MoE层是多层的一部分, 其他层补偿
- 负载均衡损失不断减少不均衡程度
- **实测: 丢弃率10-15%时质量下降<0.5%**

关键要点: 容量因子CF=1.25是最佳权衡: 内存开销增25%, 质量接近最优。Token Dropping通过残差连接保证鲁棒性, 丢弃10-15%质量几乎无损。

Fedus et al. (2021) Switch Transformers Table 1, Section 2.2 | arXiv:2101.03961

1.6 Switch训练稳定性: 精度、初始化与正则化

背景: 大规模MoE模型训练不稳定是公认难题。Switch Transformer系统性地解决了三个关键问题: (1) 混合精度训练中Router的数值不稳定; (2) 专家初始化过大导致早期训练发散; (3) Fine-tuning阶段过拟合。这些技术使Switch首次稳定训练到万亿参数规模。

问题1: bfloat16的精度陷阱

bfloat16特点: 8位指数 + 7位尾数
vs float32: 8位指数 + 23位尾数
尾数精度差: 2^(23-7) = 65536倍!

为什么Router特别敏感?

Router输出经过Softmax: exp(x)/sum(exp(x))

- exp()放大微小误差 -> 路由决策翻转
- 一个token被错误路由 -> 梯度传播错误

解决方案: 选择性混合精度

- 主体计算: bfloat16 (节省内存和计算)
- **Router: float32 (保证路由精度)**
- 损失计算: float32 (数值稳定)

精度对比 (Switch Table 2)

精度方案	质量	速度	稳定性
全float32	最佳	1.0x (慢)	稳定
全bfloat16	-	1.8x	不稳定
选择性混合精度	**接近最佳**	**1.8x**	**稳定**

问题2: 初始化过大

标准Transformer初始化: s = 1/sqrt(d_model)
Switch缩小10x: s = 0.1 / sqrt(d_model)
原因: MoE层初期路由不稳, 小初始化减缓发散

问题3: Fine-tuning过拟合

MoE参数量巨大, 小数据集极易过拟合
解决: 专家层独立Dropout = 0.4
(其他层保持原始Dropout 0.1)

- 效果: SuperGLUE从不稳定收敛到79.5分

关键要点: 三项关键技术使万亿参数MoE稳定训练: 选择性混合精度(Router用float32), 缩小初始化(s=0.1), 增大专家Dropout(0.4)。

Fedus et al. (2021) Switch Transformers Table 2, Section 3 | arXiv:2101.03961

1.6 Switch Transformer扩展结果: 从Base到万亿参数

背景: Switch Transformer通过系统性实验证明了MoE在不同规模下的扩展效果。从Base(223M)到XXL(11B Dense基准)再到万亿参数(1.6T), 各层面均展示出显著加速。特别在多语言任务中, 101种语言中91%获得超4倍加速。

预训练加速 (vs T5基准)

模型	参数量	FLOPs	加速(step)	加速(wall)
T5-Base	223M	124B	1.0x	1.0x
Switch-Base 128E	**7.4B**	124B	**7.5x**	**2.5x**
T5-Large	739M	425B	1.0x	1.0x
Switch-Large 128E	**26B**	425B	**5.8x**	**2.0x**

Fine-tuning结果

任务	T5-Base	Switch-Base	提升
SuperGLUE	75.1	**79.5**	+4.4
ANLI R3	32.2	**36.9**	+4.7
WinoGrande	65.8	**72.0**	+6.2

专家数量的扩展效果:

专家数: 8 -> 16 -> 32 -> 64 -> 128 -> 256

- 加速持续提升, 呈log-log线性关系
- 256专家仍有提升, 但边际递减

万亿参数模型:

- Switch-XXL: 395B参数, 4x加速 vs T5-XXL
- **Switch-C: 1.6T参数 (2048专家), 4x加速**
- 首次在此规模上稳定训练MoE

多语言任务 (mC4 corpus)

101种语言, 平均加速5x

- 91%的语言获得超4倍加速
- 低资源语言受益最大 (最高20x+)

知识蒸馏 (Distillation)

- 大MoE -> 小Dense: 保留37%加速
- Switch-Base -| T5-Base: 2.7x加速保留
- 解决推理阶段的内存问题

总结: Switch在所有维度超越Dense

- 预训练: 7.5x step加速, 2.5x wall加速
- 下游: SuperGLUE +4.4分
- 多语言: 5x均值加速
- 规模: 首个稳定1.6T参数模型

关键要点: Switch实现全面扩展: 7.5x预训练加速、SuperGLUE +4.4、多语言5x均值加速、1.6T参数稳定训练。MoE是Scaling最高效路径。

Fedus et al. (2021) Switch Transformers Tables 3,5,7,9 | arXiv:2101.03961

1.6 MoE架构演进对比: Shazeer -> GShard -> Switch

系统视角总结: 三个里程碑代表MoE从"学术概念"到"产业系统"的完整演化: Shazeer证明了可行性, GShard解决了分布式系统, Switch实现了极致简化。每一步都在系统设计上做出关键权衡。

特性	Shazeer MoE (2017)	GShard (2020)	Switch Transformer (2021)
路由策略	Noisy Top-K (K=4)	Top-2 + 随机第二专家	**Top-1 (最简)**
最大规模	137B参数, 65536专家	600B参数, 2048专家	**1.6T参数, 2048专家**
基础架构	LSTM	Transformer (Encoder)	Transformer (Enc-Dec)
MoE层放置	每层	每隔一层	每层或每隔一层
负载均衡	Importance Loss	辅助损失 l_aux	简化均衡损失 + CF
容量控制	无 (软约束)	CF=2.0, 硬丢弃	CF=1.25, 选择性丢弃
精度策略	float32	bfloat16	选择性混合精度
分布式方案	Model Parallelism	SPMD + Expert Parallelism	EP + DP混合
硬件	GPU集群	2048 TPU v3	TPU v3 pods
关键成果	ppl: 28.0 vs 30.6	BLEU 44.3, 4天训练	**7x加速, 1.6T稳定训练**
核心贡献	**证明MoE可扩展至千亿**	**开创MoE分布式系统**	**简化路由, 极致规模化**

关键要点: 三代MoE的演进逻辑: 可行性(Shazeer) -> 分布式系统(GShard) -> 极致简化(Switch)。每代都在系统设计上做出关键trade-off。

综合: Shazeer 2017 | Lepikhin 2020 | Fedus 2021

PART 07

MoE的系统视角

从算法到部署的关键挑战

1.7 MoE系统挑战概览: 从算法到部署

背景: MoE在算法层面看似简单(选专家、算加权和), 但在实际系统部署中面临一系列Dense模型不存在的独特挑战。这些挑战横跨通信、内存、负载均衡、硬件利用率等多个维度, 是第二、三课时的核心内容。本slide作为预览, 帮助建立全局视角。

通信

通信开销 (All-to-All)

MoE需要All-to-All通信(每对GPU都交换数据), 而Dense模型仅需AllReduce。All-to-All的通信量随路由动态变化, 难以优化。平均占训练步进时间34.1%, 多节点可达80%。

均衡

负载不均衡 (Load Imbalance)

路由天然倾向于将token集中到少数"热门"专家, 导致计算资源浪费。极端情况下发生"专家坍缩": 只有1-2个专家被使用, 其余idle。需要辅助损失、容量因子等机制缓解。

内存

内存管理 (Memory)

总参数量巨大(671B for DeepSeek-V3), 单GPU无法容纳。需要Expert Parallelism将专家分布到不同GPU。激活缓存(activation buffer)因动态路由难以预分配。

放置

专家放置 (Expert Placement)

如何将专家分配到GPU? 随机放置导致通信热点。需要考虑: 同组专家共置、负载感知放置、拓扑感知分配。DeepSeek-V3使用16路PP + 64路EP。

扩展

扩展复杂度 (Scaling)

Dense模型扩展路径清晰(DP/TP/PP), MoE增加EP维度后组合爆炸。需要同时优化4种并行策略的组合, 且最优配置因硬件拓扑而异。

关键要点: MoE的系统挑战远超算法层面: 通信(All-to-All)、负载均衡、内存管理、专家放置和多维并行。第二三课时将逐一深入。

预览: Session 2 (通信与并行) + Session 3 (优化与部署)

1.7 MoE通信模式: AllReduce vs All-to-All

系统核心: All-to-All通信是MoE区别于Dense模型的最根本系统差异。Dense模型训练中使用AllReduce聚合梯度(固定通信模式), 而MoE需要All-to-All交换token(动态通信模式)。理解两者的区别是理解MoE系统挑战的关键。

AllReduce (Dense模型)

- 用途: 数据并行中聚合梯度
- 每个GPU拥有完整模型副本
- 通信内容: 梯度 (固定大小, 可预测)
- 通信模式: Ring AllReduce, 高度优化
- 带宽利用率: 接近理论最优

All-to-All (MoE模型)

- 用途: 将token发送到对应专家所在GPU
- 每个GPU只有部分专家
- 通信内容: token (动态大小, 取决于路由)
- 通信模式: 每对GPU间交换不同量的数据
- 带宽利用率: 远低于理论值

为什么All-to-All更难优化?

- 动态路由 -> 消息大小不固定, 无法预分配buffer
- 需要容量因子预留buffer空间(浪费内存)
- 跨节点通信延迟远高于节点内

通信开销数据

场景	All-to-All占比	瓶颈
单节点 (8 GPU)	~15-20%	NVLink带宽
多节点 (常规)	**~34.1%**	IB带宽
多节点 (极端)	**~80%**	网络拓扑

通信对比总结:

特性	AllReduce	All-to-All
消息大小	固定	动态
通信模式	可预测	路由依赖
优化程度	高度优化	仍有挑战
硬件支持	NCCL原生	需要定制优化
跨节点	可overlap	**瓶颈严重**

解决方向 (Session 2详述):

- 通信与计算重叠 (overlap)
- 限制专家放置拓扑(节点内EP)
- 压缩通信: 量化token特征

关键要点: All-to-All通信是MoE最大系统瓶颈: 动态消息+跨节点延迟, 占步进时间34%-80%。Dense模型的AllReduce完全不存在这一问题。

数据综合: Hwang et al. 2023, Rajbhandari et al. (DeepSpeed-MoE) 2022

1.7 专家并行 (Expert Parallelism): MoE的第四维并行

背景: Dense模型有三种经典并行策略: 数据并行(DP)、张量并行(TP)、流水线并行(PP)。MoE引入第四种: 专家并行(EP) -- 将不同专家放在不同GPU上。生产系统通常混合使用四种策略, 例如DeepSeek-V3: 16路PP + 64路EP。

四种并行策略对比

特性	数据并行 (DP)	张量并行 (TP)	流水线并行 (PP)	专家并行 (EP)
切分对象	数据 (batch)	权重矩阵	模型层	专家网络
通信类型	AllReduce	AllReduce	P2P (send/recv)	All-to-All
通信频率	每步1次	每层2次	每micro-batch	每MoE层2次
最佳放置	跨节点	节点内 (NVLink)	跨节点	灵活配置
内存优势	梯度分摊	权重分摊	层分摊	**专家分摊**
适用场景	所有模型	大矩阵运算	深层模型	**MoE模型**

生产系统的混合并行方案

系统	注意力层	MoE层	总GPU	关键设计
GShard	DP (数据并行)	EP (专家并行)	2048 TPU	SPMD, 同一程序
Switch	DP + MP	EP	TPU pods	每层灵活配置
DeepSeek-V3	DP + 16路PP	64路EP	2048 H800	PP+EP混合, 通信overlap

EP的核心权衡: EP度越高, 每GPU专家越少(内存更轻), 但All-to-All通信越多。节点内EP利用NVLink(~900GB/s), 跨节点EP受限于IB(~400Gb/s)。最优EP度取决于硬件拓扑。

> **关键要点:** EP是MoE独有的第四维并行: 将专家分布到不同GPU。生产系统混合DP+TP+PP+EP四种策略, 最优组合取决于硬件拓扑。

GShard: Lepikhin 2020 | Switch: Fedus 2021 | DeepSeek-V3: arXiv:2412.19437

1.7 MoE内存与计算分析: 稀疏计算的代价

系统分析: MoE模型的内存和计算特征与Dense模型截然不同。Dense模型是"计算密集型": 算力是瓶颈。MoE模型是"通信密集型": All-to-All通信和内存管理是瓶颈。理解这一差异是优化MoE系统的基础。DeepSeek-V3的计算/通信比接近1:1, 说明通信已成为主要矛盾。

内存组成分析

1. 专家权重 (Expert Weights)

- 总量巨大: DeepSeek-V3 = 671B参数
- 通过EP分布式存储: 每GPU仅存部分专家
- 64路EP: 每GPU约 671B/64 ~ 10.5B参数

2. 注意力层权重 (Replicated)

- 注意力层在所有EP设备上复制
- 占总参数量较小但必须在每GPU存储
- 可通过TP进一步切分

3. 激活缓存 (Activation Buffer)

- All-to-All需要预分配收发buffer
- 大小 = batch_size * CF * d_model * 2 (收+发)
- CF越大, buffer越大, 内存浪费越多
- 动态路由使buffer大小难以精确预估

计算特征分析

Dense模型: 计算密集型

FLOPs/token = 2N (N为总参数)

- GPU利用率可达40-60% MFU
- 通信可与计算重叠

MoE模型: 通信密集型

FLOPs/token = 2 * K * expert_size (K<<N)

- 单专家计算量小, GPU利用率低
- 通信难以完全与计算重叠

DeepSeek-V3的关键指标:

指标	数值
总参数	671B
激活参数	37B (5.5%)
计算/通信比	~1:1
训练token数	14.8T
训练成本	$5.58M (2048 H800)

关键要点: MoE从"计算密集型"转为"通信密集型": DeepSeek-V3计算/通信比~1:1。优化MoE的关键不是加速计算, 而是减少通信。

本课小结: MoE基础原理与经典架构

第一课时覆盖了MoE的动机、核心公式、发展历程和三大经典架构。 从Scaling Laws出发，理解"为什么需要MoE"，再到公式、组件和系统挑战，建立了对MoE的全局理解。

核心概念回顾

1. Scaling Laws与Dense困境

- 参数量是最有效的缩放维度，但Dense模型中 C = 2N
- 需要: 增加模型容量而不增加计算量

2. MoE核心公式

- y = Sum G(x)_i * E_i(x)，大部分G(x)_i = 0
- N个专家只激活K个，FLOPs与N无关

3. 发展历程: 1991 -> 2025

- Jacobs -> Jordan -> Shazeer -> GShard -> Switch -> 产业化

4. 三大经典架构

- Shazeer 2017: Noisy Top-K，137B参数，证明可行性
- GShard 2020: Top-2，600B，SPMD+EP，开创系统范式
- Switch 2021: Top-1，1.6T，7x加速，极致简化

5. 系统挑战 (预览)

- All-to-All通信 | 负载均衡 | 内存管理 | 专家放置 | 多维并行

第二课时预告

MoE的系统优化与高级架构

负载均衡深入

- 专家坍缩问题及其解决方案
- Expert Choice路由: 让专家选token
- Router Z-Loss (ST-MoE)

高级路由与架构

- DeepSeek-V3: 辅助损失-free均衡
- Shared Expert + Routed Expert混合
- Fine-grained vs Coarse-grained MoE

分布式训练实战

- EP+DP+PP+TP四维混合并行
- 通信-计算重叠策略
- Megablock与GroupGEMM优化

关键要点: 第一课时建立了MoE全局认知: 动机(Scaling Laws) -> 公式(稀疏加权) -> 架构(三代演进) -> 系统挑战(通信为核心矛盾)。

推荐阅读: MoE核心论文与学习资源

以下列出本课涉及的核心论文和推荐资源, 按时间顺序排列。建议优先阅读带星号(*)的论文。

核心论文 (必读)

[1] Kaplan et al. (2020) "Scaling Laws for Neural Language Models"
arXiv:2001.08361 -- 揭示参数/数据/计算的幂律关系

[2] * Jacobs et al. (1991) "Adaptive Mixtures of Local Experts"
Neural Computation 3(1) -- MoE原始论文

[3] * Shazeer et al. (2017) "Outrageously Large Neural Networks"
ICLR 2017, arXiv:1701.06538 -- 首个大规模MoE, Noisy Top-K门控

[4] * Lepikhin et al. (2020) "GShard: Scaling Giant Models"
arXiv:2006.16668 -- 首个MoE Transformer系统, Expert Parallelism

[5] * Fedus et al. (2021) "Switch Transformers: Scaling to Trillion Parameters"
JMLR 2022, arXiv:2101.03961 -- Top-1路由, 1.6T模型, 7x加速

[6] Zoph et al. (2022) "ST-MoE: Designing Stable and Transferable Sparse Expert Models"
arXiv:2202.08906 -- Router Z-Loss, 训练稳定性最佳实践

[7] Zhou et al. (2022) "Mixture-of-Experts with Expert Choice Routing"
NeurIPS 2022 -- 反转路由: 专家选token, 天然负载均衡

核心论文 (续) + 技术报告

[8] * DeepSeek-AI (2024) "DeepSeek-V3 Technical Report"
arXiv:2412.19437 -- 671B/37B active, 辅助损失-free均衡

[9] Cai et al. (2025) "A Survey on Mixture of Experts"
arXiv:2503.07137 -- 最新综合MoE综述 (截至2025.3)

[10] Hoffmann et al. (2022) "Training Compute-Optimal LLMs" (Chinchilla)
arXiv:2203.15556 -- 修正Scaling Laws, 数据与模型等比缩放

推荐博客与教程

[A] Maarten Grootendorst "A Visual Guide to MoE"
newsletter.maartengrootendorst.com -- 最佳MoE可视化教程

[B] HuggingFace "Mixture of Experts Explained"
huggingface.co/blog/moe -- 代码示例, 实践指南

[C] NVIDIA Technical Blog "Applying Mixture of Experts"
developer.nvidia.com/blog -- Megatron-LM中的MoE实现

关键要点: 优先阅读带*的5篇核心论文: Shazeer 2017(算法), GShard 2020(系统), Switch 2021(规模), DeepSeek-V3(前沿), Jacobs 1991(起源)。

第二课时

路由机制、训练优化与系统设计

Token Choice vs Expert Choice | 负载均衡 | 专家坍缩 | 分布式训练

回顾: 第一课时核心概念

架构	路由	规模	关键创新
Shazeer	Top-4	137B	Noisy门控+负载均衡
GShard	Top-2	600B	Expert Parallelism
Switch	Top-1	1.6T	简化路由, Capacity Factor

MoE核心公式与关键思想

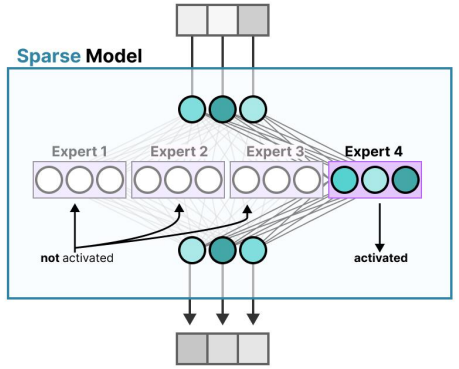
- y = Sum G(x)_i * E_i(x) -- 稀疏门控的加权求和
- 条件计算: 不同输入激活不同专家, 参数量与计算量解耦
- Top-K稀疏性: FLOPs仅取决于K, 与总专家数N无关

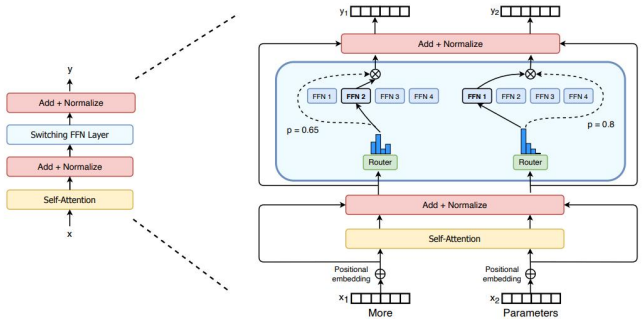
经典架构演进

- Shazeer 2017: 首个大规模MoE, 137B参数, Noisy Top-K门控
- GShard 2020: 600B参数, Expert Parallelism, All-to-All通信, 4天训练
- Switch 2021: Top-1路由简化, 1.6T参数, 7x预训练加速

本节课关注的问题

- 路由算法: 除了Top-K, 还有哪些路由策略? 各有什么优劣?
- 负载均衡: 如何防止专家坍缩? Auxiliary Loss够用吗?
- 系统设计: Expert Parallelism如何与DP/TP/PP结合? 通信如何优化?





第一课时建立了MoE的基本概念。本课时深入路由机制、训练优化和系统设计 -- MoE从"能用"到"好用"的关键。

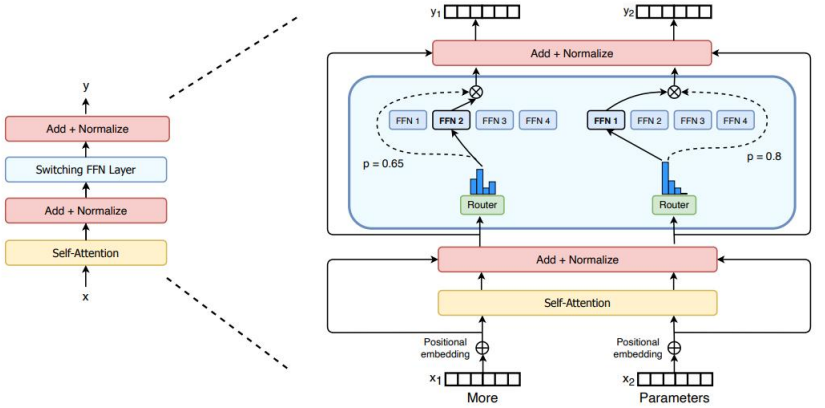
PART 01

路由算法全景

Token Choice | Expert Choice | Hash | Sinkhorn | Dynamic Routing

2.1 Token Choice (Top-K): 标准路由方法

背景: Token Choice是最经典的路由方式 -- 每个token独立地从N个专家中选择Top-K个。从Shazeer 2017开始使用，至今仍是大多数MoE模型的基础。它的优势是实现简单，但天然存在负载不均衡问题。



工作流程

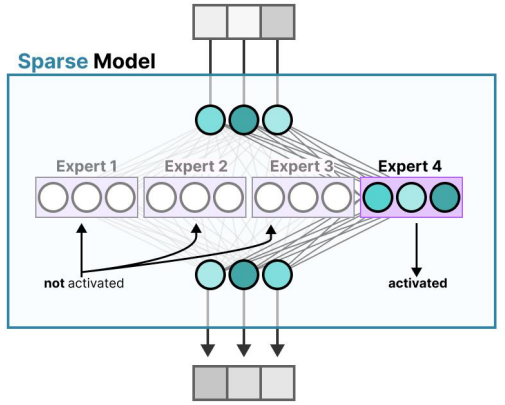
- 1. Router线性层: token向量 x -> N个logits
- 2. Softmax归一化: logits -> 概率分布
- 3. Top-K选择: 保留概率最高的K个专家
- 4. 加权求和: 输出 = Sum(gate_i * expert_i(x))

主要模型的路由选择

- GShard: Top-2, 第二个专家随机路由(概率正比于gate权重)
- Switch Transformer: Top-1, 通信量减半, 质量不降
- Mixtral 8x7B: Top-2 from 8专家, 47B参数仅12.9B激活

核心问题: 负载不均衡

- 热门专家被大量token涌入, 冷门专家几乎闲置
- Over-capacity比例可达20-40% (部分token被丢弃)
- 需要auxiliary loss来缓解, 但调参困难(太强损害性能, 太弱无效)



Token Choice实现简单但天然不均衡。关键trade-off: auxiliary loss太强损害主任务, 太弱则不均衡。这催生了Expert Choice等新方法。

Shazeer 2017 | GShard 2020 | Switch 2021 | Mixtral arXiv:2401.04088

2.1 Expert Choice: 让专家选择token

核心思想: 如果token选专家导致负载不均, 何不反过来让专家选token? Expert Choice (Zhou et al., NeurIPS 2022) 正是这样做的 -- 每个专家从全部token中选择自己要处理的top-k个, 天然保证每个专家处理相同数量的token。

算法流程

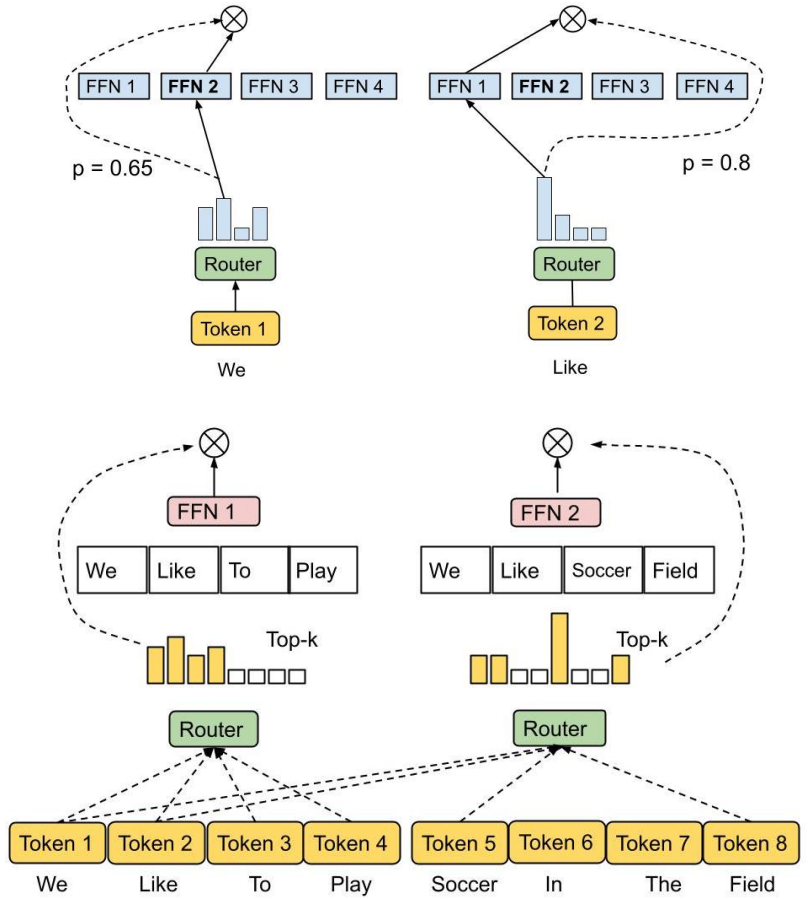
- 1. 计算token-expert亲和度矩阵 (与Token Choice相同)
- 2. 每个专家从矩阵中选择亲和度最高的k个token
- 3. 专家处理选中的token, 加权返回结果

关键优势

- 天然负载均衡: 每个专家处理固定数量token, 无需auxiliary loss
- 2x训练加速: 在8B/64E模型上, 比Switch/GShard收敛快2倍
- 灵活专家分配: 不同token可获得0到多个专家(复杂token更多计算)

局限性

- 部分token可能被0个专家选中 -> 信息丢失(通过residual缓解)
- 推理时需要全局token信息, 不利于自回归解码
- 实现复杂度高于简单Top-K



Expert Choice通过反转路由方向实现天然负载均衡, 训练速度提升2倍。但推理时的全局依赖限制了它在自回归模型中的直接应用。

Zhou et al., 'Mixture-of-Experts with Expert Choice Routing', NeurIPS 2022 (arXiv:2202.09368)

2.1 Hash Routing与Sinkhorn Routing

除了Token Choice和Expert Choice，还有两类不同思路的路由方法：基于哈希的确定性方法和基于最优传输的均衡方法。

Hash Routing (确定性路由)

expert = token_id % N (简单取模)

- 无可学习参数，无auxiliary loss，零路由开销
- 完美负载均衡 (每个专家获得等量token)
- 但：分配完全无视token语义 -> 专家学习重叠表示
- 性能显著差于learned routing

Sinkhorn Routing (最优传输)

用Sinkhorn算法求解token-expert最优匹配

- 交替归一化分配矩阵的行和列 -> 均衡分配
- Hash级别的负载均衡 + 学习能力
- 关键问题：Sinkhorn迭代会截断梯度
- 解决：用Sinkhorn选专家，用原始logits计算权重

Cerebras Router Wars 实验对比 (Soboleva, Dec 2025)

- 128专家时：Learned和Sinkhorn路由比Hash路由质量提升3倍
- 16专家时：Learned路由的loss改善比Hash大3倍 (4% vs ~1.5%)
- 专家数越多，Learned routing的优势越明显
- Hash路由虽然均衡完美，但专家学到的表示高度重叠 -> 本质上退化为Dense
- **结论：当前所有生产MoE都使用某种形式的learned routing + 工程tricks**

Hash简单但性能差，Sinkhorn均衡但截断梯度。实践中：learned routing + auxiliary loss仍是主流，Expert Choice是最有前景的替代。

Roller 2021 NeurIPS | Lewis 2021 ICML | Cerebras 'Router Wars' blog Dec 2025

2.1 Dynamic Routing与路由算法总结

Dynamic Routing的核心问题: 为什么每个token都要选固定K个专家? 简单token可能只需1个专家, 而复杂token可能需要更多。Huang et al. (ACL 2024)提出按累积概率动态选择专家数量。

Dynamic Routing (Huang et al., ACL 2024)

- 如果最高专家概率 > 阈值p: 仅激活1个专家
- 否则逐个加入专家, 直到累积概率超过阈值p
- 实验: p=0.4, 平均激活1.76个专家(少于固定Top-2), 性能提升0.7%+
- 后续: DTop-p (Jin 2025) 用PI控制器动态调整阈值

五种路由算法全面对比

算法	负载均衡	专精能力	需Aux Loss	可变专家/token	实现复杂度	代表模型
Token Choice(Top-K)	差(需aux loss)	高(中间层)	是	否(固定K)	低	GShard/Switch/Mixtral
Expert Choice	完美(保证)	好	否	是(0到多)	中	Google EC 2022
Hash Routing	完美	差(无语义)	否	否	极低	BASE Layers
Sinkhorn	近完美	中等	否(截断)	否	中高	S-BASE
Dynamic	中等	高	是(+熵正则)	是(自适应)	中	ACL 2024

没有完美的路由算法。Token Choice最成熟, Expert Choice最均衡, Dynamic最灵活。实际选择取决于模型规模和部署需求。

Huang et al. ACL 2024 | Jin et al. 2025 | Cerebras 'Router Wars' 2025

PART 02

负载均衡: 算法与系统视角

Auxiliary Loss | Router Z-Loss | Loss-Free Balancing

2.2 Auxiliary Loss: 最经典的负载均衡方法

为什么需要负载均衡? 没有负载均衡的MoE会退化 -- 门控网络倾向于将所有token发给少数"明星专家", 其他专家几乎不被使用。这是一个自增强循环: 热门专家获得更多训练 -> 变得更好 -> 被更多token选中。Auxiliary Loss通过在损失函数中加入均衡惩罚来打破这个循环。

Shazeer 2017: Importance Loss + Load Loss

Importance(X) = Sum_x G(x) (每个专家的总gate权重)
L_importance = w * CV(Importance)^2
CV = 标准差/均值 (变异系数, 衡量分布均匀程度)

Switch Transformer简化版

L_aux = alpha * N * Sum(f_i * P_i)
f_i = token分配给专家i的比例, P_i = router概率分配给专家i的比例
alpha = 10^-2 (扫过10^-1到10^-5, 10^-2最佳)

核心trade-off (MoE训练中最棘手的调参问题)

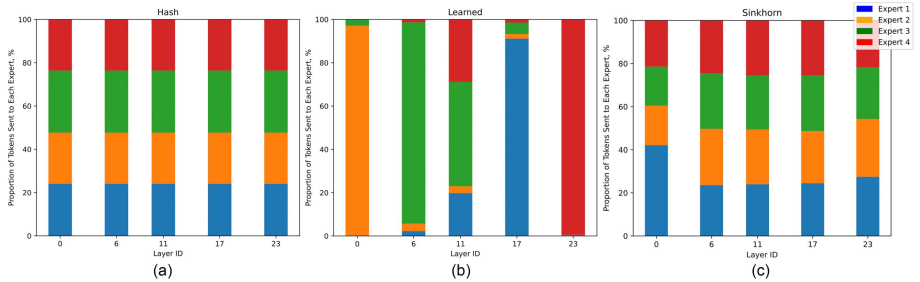
- alpha太大: auxiliary loss压过语言模型目标, 专家过度均匀化, 丧失专精能力
- alpha太小: 负载均衡效果不足, 仍然存在专家坍缩
- 不同层/不同训练阶段可能需要不同的alpha, 但目前都用统一值

Shazeer 2017 实验数据 (Table 6)

w_imp	w_load	Perplexity	Max/Mean Load
0	0	39.8	17.80
0.1	0	35.6	1.64
0.1	0.1	35.6	1.14

解读:

- 无均衡: perplexity 39.8, 最繁忙专家负载是平均的17.8倍!
- 加入均衡: perplexity降至35.6, 负载比降至1.14 (接近完美均衡)
- **负载均衡不仅改善了均匀性, 还提升了模型质量!**



Auxiliary Loss是MoE负载均衡的基础方法, 但存在alpha调参困难的根本问题。DeepSeek提出的Loss-Free方法(后续)尝试彻底解决这个问题。

Shazeer 2017 (Table 6) | Fedus 2021 Switch Transformer | Cerebras Router Wars 2025

2.2 Router Z-Loss: 解决训练不稳定

问题背景: 即使有了Auxiliary Loss, 大规模MoE模型在训练过程中仍然频繁出现loss突然发散(spike)甚至NaN。ST-MoE (Zoph et al., 2022) 发现根因在于router logits过大导致的数值不稳定 -- 特别是在bfloat16精度下, 大logits经过softmax的指数运算后会产生严重的舍入误差。

Router Z-Loss公式 (Zoph et al., 2022)

$$L_z = (1/B) * \text{Sum\_i } (\log \text{Sum\_j } \exp(x_{ij}))^2$$

B = batch size, N = 专家数, x_ij = router对第i个token第j个专家的logit

为什么有效?

- 对log-sum-exp取平方 -> 二次惩罚: LSE=10贡献100, LSE=20贡献400
- 强力约束logits大小, 防止指数爆炸
- bfloat16的舍入误差比float32大65536倍, Z-Loss防止这个问题

总损失函数

$$L_{total} = L_{CE} + c_b * L_{balance} + c_z * L_z$$

- **关键性质: 稳定训练的同时不损害质量(有时甚至略有提升)**
- 对比: 直接clip logits会损害性能, Z-Loss是软性约束更优

ST-MoE关键成果

- 269B稀疏参数 = 32B Dense的计算量
- 在SuperGLUE, ARC, XSum等基准上达到SOTA
- **训练不再出现irrecoverable loss spikes**

MoE层放置建议:

- Top-2路由(非Top-1)
- Capacity Factor = 1.25 (训练)
- 每隔一层替换FFN为MoE
- 最多1个专家/TPU core

Router Z-Loss通过软性约束logits大小解决了bfloat16精度下MoE训练不稳定的问题。它与Auxiliary Loss互补, 二者一起使用效果最佳。

Zoph et al., 'ST-MoE: Designing Stable and Transferable Sparse Expert Models', 2022 (arXiv:2202.08906)

2.2 Loss-Free Balancing: DeepSeek的突破

核心问题: Auxiliary Loss的根本困境在于它会引入干扰梯度 -- 均衡目标和语言模型目标是两个不同的优化方向，它们共享同一个loss函数意味着彼此干扰。DeepSeek提出了一种完全不使用auxiliary loss的均衡方法，在ICLR 2025被接收。

Loss-Free Balancing算法

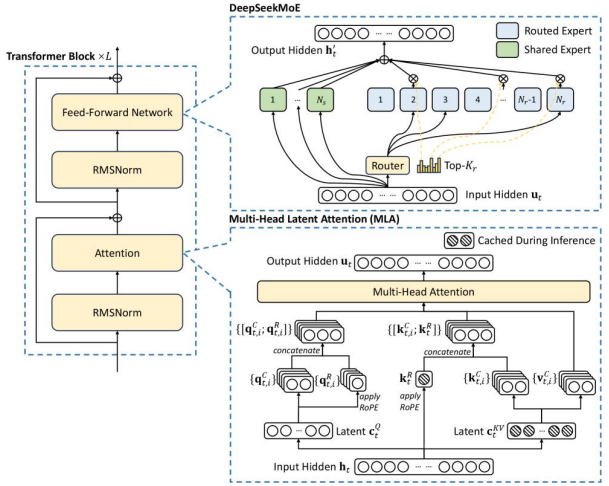
在Top-K路由决策前，对每个专家的routing score加上一个bias:

b_i = b_i + u * sign(e_i)

u = 更新速率, e_i = 负载违反量(该专家实际负载 - 平均负载)

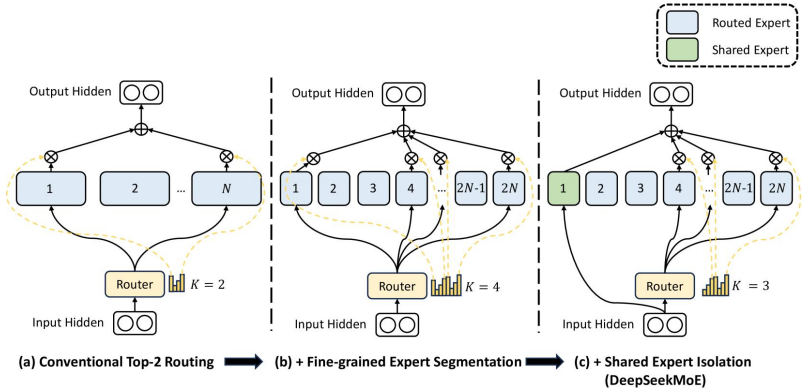
关键设计

- Bias只用于路由决策(选谁)，不用于计算门控值(权重多少)
- **因此: 零干扰梯度! 语言模型目标的梯度完全不受均衡影响**
- 使用Sigmoid门控(非Softmax)，实验发现效果更好
- DeepSeek-V3: gamma=0.001 前14.3T tokens，然后0.0 后500B tokens



实验结果 (Wang et al., ICLR 2025)

模型	方法	Perplexity	MaxVio
1B	Aux Loss	9.56	0.72
1B	Loss-Free	9.50	0.04
3B	Aux Loss	7.97	0.52
3B	Loss-Free	7.92	0.04



Loss-Free Balancing通过将均衡与优化完全解耦，在更好的perplexity下实现了MaxVio从0.72降至0.04。DeepSeek-V3的14.8T token训练验证了其稳定性。

Wang et al., 'Auxiliary-Loss-Free Load Balancing', ICLR 2025 (arXiv:2408.15664) | DeepSeek-V3 arXiv:2412.19437

PART 03

训练稳定性与专家坍缩

MoE训练中最棘手的问题

2.3 专家坍缩: MoE最大的失败模式

专家坍缩(Expert Collapse)是MoE训练中最具破坏性的问题 -- 多个专家的参数趋同, 学到几乎相同的函数, 使得MoE退化为一个昂贵的Dense模型。它让你付出了N个专家的内存代价, 但只获得了1个专家的性能。

两种坍缩形式

1. 参数坍缩 (Expert Collapse)

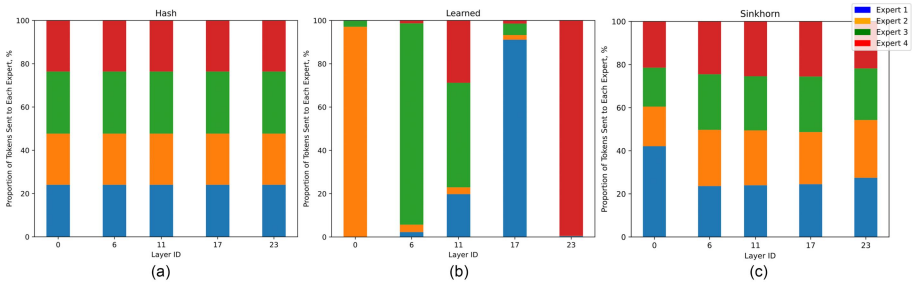
- 多个专家的权重矩阵趋于相同
- 诊断: 计算专家权重的两两余弦相似度, 高相似度 = 坍缩
- 原因: 初始化相似 + 相似token分配 -> 相似梯度更新

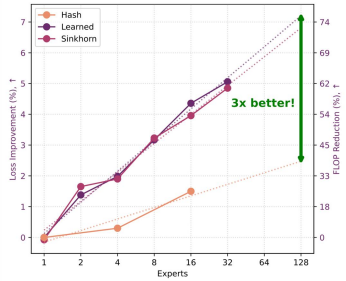
2. 路由坍缩 (Routing Collapse)

- 自增强循环: 热门专家 -> 更多训练 -> 更好 -> 更热门
- 其他专家几乎不被使用, 逐渐退化
- 一旦形成, 极难逆转 (路由模式在训练早期固化)

OpenMoE重要发现

- 路由主要由token ID决定(而非上下文!) -- 'Context-Independent Specialization'
- 路由模式在预训练早期(前10%steps)就基本固化
- 后续训练和微调几乎无法改变路由分配





Cerebras实验: Learned路由在早期/晚期层出现严重坍缩
(大部分token被发送给1-2个专家, 其余闲置)

Cerebras: 'Router能单独毁掉你的模型 -- 架构完美、超参调好、算力充足, 但router坍缩就退回Dense性能。'

2.3 防止专家坍缩的六种策略

专家坍缩没有银弹解决方案，但以下策略的组合可以显著降低风险。

1. 噪声注入 (Shazeer 2017)
在router logits中加入可学习高斯噪声，迫使模型在训练初期探索不同专家组合，防止过早锁定。噪声量由可学习参数W_noise控制，训练后期自然减小。

2. Cosine Router
对token表示和专家嵌入都做L2归一化，使相似度计算更稳定。防止某些专家因为嵌入范数更大而垄断路由，缓解表示坍缩。

3. 差异化初始化
用不同随机种子初始化各专家，或使用Hadamard矩阵强制正交初始化。确保专家从一开始就有不同的参数，而非从相同起点出发。

4. 共享专家 (DeepSeek-V3, Qwen2)
设置若干always-activated的共享专家来处理通用知识(如英语语法)，释放routed专家去学习专精领域。减轻了routed专家必须学习公共知识的压力。

5. 专精化损失
惩罚同一层内不同专家对相同输入产生过于相似的激活。这是一个即插即用的auxiliary loss，不需要修改router或模型架构。

6. Loss-Free Balancing (DeepSeek)
动态bias避免了auxiliary loss的梯度干扰问题。DeepSeek-V3在14.8T token上实现了'remarkably stable'的训练，没有出现irrecoverable loss spikes。

实践中通常组合使用: 噪声注入 + Auxiliary Loss + Router Z-Loss (基础配置), 或 Loss-Free Balancing + 共享专家 (DeepSeek方案)。

Shazeer 2017 | Zoph 2022 | DeepSeek-V3 2024 | Wang ICLR 2025

PART 04

细粒度专家与共享专家

DeepSeekMoE: 走向极致的专家专精

4.1 知识冗余与知识混杂问题 (Knowledge Redundancy & Hybridity)

背景: 传统MoE (如Mixtral的8专家架构) 在扩展时面临两个根本性问题, 这些问题直接限制了MoE的参数效率和专家利用率。DeepSeek团队深入分析了这些问题并提出了针对性解决方案。

问题一: 知识冗余 (Knowledge Redundancy)

- 现象: 多个专家学习相同的通用知识 (common knowledge)
- 根因: 每个专家独立初始化, 缺乏知识分工机制
- 后果: 大量参数被浪费在重复存储通用知识上
- 类比: 8个全科医生 vs 1个全科+7个专科
- 实验证据: 分析专家权重相似度, 发现40-60%参数高度冗余
- 浪费比例: 传统8专家Top-2, 有效参数利用率仅约60%
- 影响: 模型总参数量的增长并未带来等比的性能提升
- Mixtral 8x7B: 46.7B参数, 12.9B激活, 仍有大量冗余

问题二: 知识混杂 (Knowledge Hybridity)

- 现象: 有限专家需覆盖多样化领域, 无法深入专精
- 根因: 专家数量有限 (通常8-16个), 而知识领域无限
- 后果: 每个专家被迫混合处理多种不相关任务
- 类比: 让一个医生同时看内科、外科、牙科
- 表现: 路由分析发现单个专家覆盖的token类型极为分散
- 质量上限: 专家无法为特定任务学习精细化表示
- 对比: 256个细粒度专家可实现更清晰的知识边界
- 这一问题在模型规模增大时更加突出

关键要点: 知识冗余浪费参数, 知识混杂限制专精 -- 这两个问题共同驱动了DeepSeek的细粒度专家分割与共享专家隔离设计。

Dai et al. (2024) DeepSeekMoE: Towards Ultimate Expert Specialization | arXiv:2401.06066

4.2 DeepSeekMoE架构: 细粒度分割与共享专家隔离

核心思想: DeepSeekMoE通过两项创新同时解决知识冗余与混杂问题: (1) 将N个专家细分为mN个更小专家, 激活mK个, 总计算量不变但专精度大幅提升; (2) 设置K_s个常驻共享专家 (Shared Experts), 专门存储通用知识, 释放路由专家的专精空间。

创新一: 细粒度专家分割 (Fine-Grained Segmentation)

- 原始: N个专家, 每个FFN维度d_ffn, 激活Top-K
- 分割: mN个专家, 每个FFN维度d_ffn/m, 激活Top-mK
- **计算量不变: mK * (d_ffn/m) = K * d_ffn**
- 组合灵活性: C(mN, mK) >> C(N, K), 更精细的路由
- 例: 8专家Top-2 → 64专家Top-16, 组合数从28增至约1.8*10^12

创新二: 共享专家隔离 (Shared Expert Isolation)

- K_s个专家设为共享, 所有token始终经过
- 共享专家: 存储通用语言知识 (语法、常识等)
- 路由专家: 释放后可专注领域特定知识
- **输出: h = Sum(shared) + Sum(gated_routed)**
- 效果: 消除路由专家间的通用知识冗余

DeepSeek系列MoE配置

模型	总参数	激活参数	共享	路由	Top-K	层数
DeepSeekMoE 16B	16.4B	2.8B (18%)	2	64	Top-6	28
DeepSeek-V2	236B	21B (9%)	2	160	Top-6	60
DeepSeek-V3	671B	37B (6%)	1	256	Top-8+1	61

- **性能对比: DeepSeekMoE 16B (2.8B激活) 性能可比LLaMA2 7B (~2.5x激活参数)**
- DeepSeek-V3: 671B/37B参数, 256路由+1共享, Top-8+1, 61层MoE
- 趋势: 更多更小专家 + 更高稀疏度 = 更好的参数效率

关键要点: 细粒度分割提升组合灵活性, 共享专家隔离消除冗余 -- DeepSeekMoE用18%参数激活率达到传统模型性能, V3进一步将比例压至6%。

Dai et al. (2024) DeepSeekMoE | DeepSeek-AI (2024) DeepSeek-V3 Technical Report

4.3 ST-MoE: MoE设计最佳实践 (Stable and Transferable MoE)

背景: ST-MoE (Zoph et al. 2022) 是Google对MoE设计选择的系统性研究，通过大规模实验总结出一套实用的设计准则。该工作训练了269B稀疏模型，在多项任务上匹配32B稠密模型的性能，并首次系统研究了MoE的迁移学习特性。

关键设计推荐

- 路由策略: Top-2优于Top-1，兼顾性能与稳定性
- 容量因子: 训练CF=1.25，推理CF=2.0 (允许更多冗余)
- Router Z-Loss: 额外正则项，稳定门控logits

```
L_z = (1/B) * Sum(log(Sum(exp(x_i))))^2
```

惩罚过大logits，防止路由坍塌

- MoE层放置: 每隔一层替换FFN (非每层都替换)

全替换性能提升不明显但成本翻倍

- 专家数量: 128或256专家，过多会增加路由噪声
- Dropout: 专家内部0.1，路由层无dropout
- 初始化: 路由器Xavier初始化效果最佳

大规模实验结果

- **269B稀疏模型: 匹配32B稠密模型性能**
- 激活参数: 仅32B，训练FLOPs大幅降低
- SuperGLUE: 90.3 (32B dense) vs 90.1 (269B sparse)
- SQuAD: 类似精度但训练速度提升3-4x

迁移学习发现 (Transferability)

- Fine-tuning时MoE优势缩小但仍然存在
- 预训练 -> 微调: 稀疏模型gap从8x缩小到2x
- 多任务微调效果优于单任务微调
- 建议: 增大微调学习率，延长微调步数
- Expert pruning: 移除30%专家后性能仅下降2%

关键要点: ST-MoE提供了MoE设计的实用手册: Top-2路由 + CF=1.25 + Router Z-Loss + 隔层放置，269B稀疏模型匹配32B稠密性能。

PART 05

MoE分布式训练系统

Expert Parallelism · All-to-All · 混合并行

5.1 专家并行 (Expert Parallelism, EP)

核心问题: 当模型拥有数百甚至数千个专家时，单个GPU无法容纳所有专家。专家并行 (EP) 将不同专家放置在不同GPU上，通过All-to-All通信实现token的跨设备调度。这是MoE特有的并行范式，与传统的DP/TP/PP有本质区别。

EP定义与工作流

- 核心思想: 每个GPU持有专家的一个子集 (而非模型的一个切片)
- 例: 256个专家，64个GPU → 每GPU持有4个专家

Step 1 - Gate Routing (门控路由):
每个GPU上本地计算所有token的路由决策
确定每个token应发往哪个GPU的哪个专家

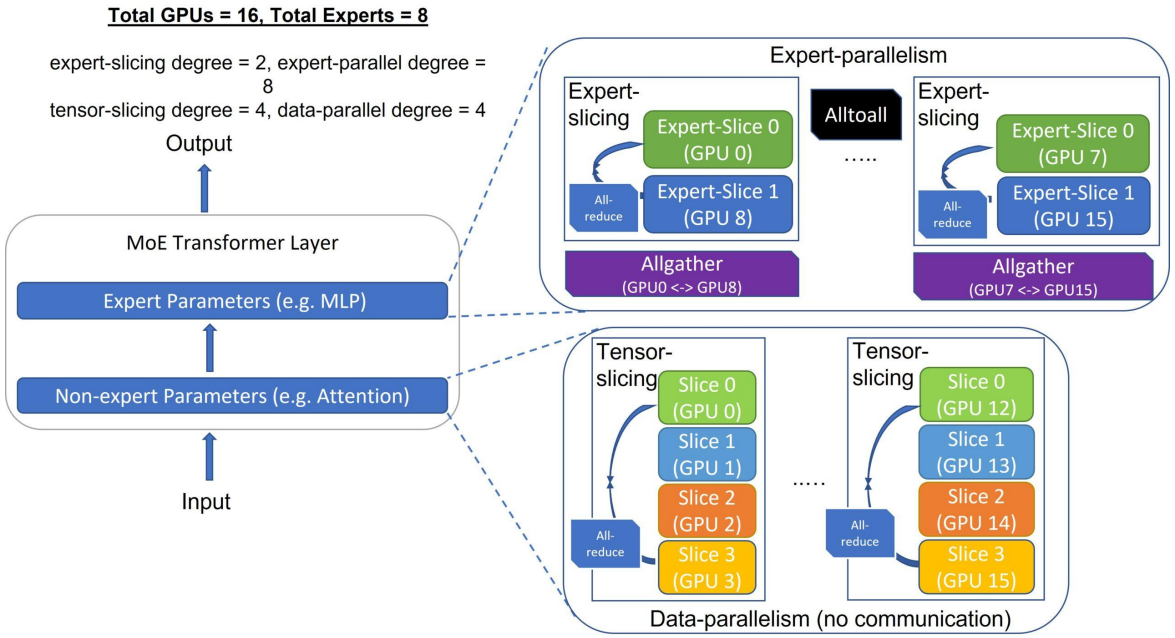
Step 2 - All-to-All Dispatch (分发):
将token通过All-to-All通信发送到目标GPU
每个GPU接收来自所有其他GPU的token

Step 3 - Expert Compute (专家计算):
各GPU上的专家并行处理接收到的token
本地计算，无需跨设备通信

Step 4 - All-to-All Combine (聚合):
专家输出通过反向All-to-All返回原始GPU
加权求和得到最终MoE层输出

通信开销分析:

- **All-to-All占训练步时间的~34.1% (单节点NVLink)**
- 跨节点 (InfiniBand/RoCE) 时可达80%以上
- 这是MoE训练的主要瓶颈，也是后续优化的核心靶点



关键要点: EP是MoE特有的并行范式，其核心挑战在于All-to-All通信 -- 单节点占34%时间，跨节点可达80%，是MoE训练的首要瓶颈。

Fedus et al. (2022) Switch Transformers | Lepikhin et al. (2020) GShard

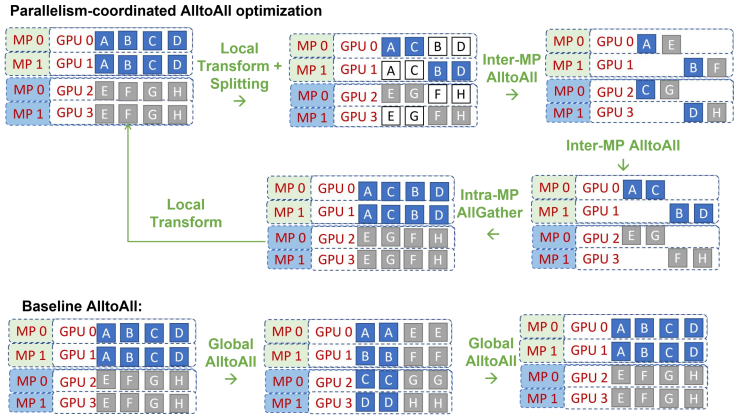
5.2 四种并行范式对比: DP vs TP vs PP vs EP

背景: 分布式训练有四种基本并行维度, MoE系统的独特之处在于引入了专家并行 (EP), 需要与其他三种范式协同工作。理解它们的区别是设计MoE训练系统的基础。

维度	数据并行 (DP)	张量并行 (TP)	流水线并行 (PP)	专家并行 (EP)
切分对象	数据batch	层内权重矩阵	层间 (层组)	专家子集
通信类型	AllReduce梯度	AllReduce激活	P2P激活	All-to-All Token
通信频率	每步1次	每层2次	每micro-batch	每MoE层2次
通信量	O(参数量)	O(batch*hidden)	O(batch*hidden)	O(batch*hidden)
最佳放置	跨节点	节点内NVLink	跨节点	节点内 (优先)
显存收益	优化器状态 (ZeRO)	权重+激活	权重	专家权重
适用模型	通用	大hidden_dim	深层模型	MoE专用

MoE训练的关键洞察:

- EP是MoE的主要扩展维度, TP通常保持1-2 (MoE单专家较小)
- Attention层: 使用DP或TP; MoE层: 使用EP
- All-to-All与AllReduce可以重叠 (overlap), 是性能优化的关键
- **DeepSeek-V3配置: 16-way PP + 64-way EP + ZeRO-1 DP, TP=1**



关键要点: MoE训练需要四维并行协同: EP是核心扩展维度, TP可保持最小, All-to-All与其他通信的重叠是优化关键。

Rajbhandari et al. (2022) DeepSpeed-MoE | DeepSeek-V3 Technical Report (2024)

5.3 生产级混合并行系统

背景: 真实的MoE训练系统从未单独使用一种并行策略，而是将多种并行维度组合成混合并行方案。不同系统的组合方式反映了硬件拓扑、模型结构和工程取舍的权衡。

系统	并行策略	模型规模	硬件	关键创新
GShard (2020)	Attn=DP + MoE=EP	600B, 2048专家	2048 TPU v3	XLA SPMD编译
Switch (2022)	EP + DP	1.6T, 2048专家	TPU v3 pods	Top-1简化通信
DeepSpeed-MoE	EP+DP+PP+TP+ZeRO	52B~4.5T	128 A100	5维并行, PR-MoE
DeepSeek-V3	16PP+64EP+ZeRO-1	671B, 257专家	2048 H800	DualPipe双向流水线
Snowflake Arctic	EP+DP+TP	480B, 128专家	多节点A100/H100	Dense+MoE residual

关键设计模式:

- 节点内优先NVLink: EP和TP放在节点内 (高带宽)
- 节点间用PP和DP: 通信量相对较小，可容忍高延迟
- DualPipe (DeepSeek-V3): 双向流水线，计算与通信完全重叠
  前向和后向pass在流水线中双向流动，消除气泡
- DeepSpeed-TED: 基于拓扑感知的EP+DP映射，40B MoE在128 V100上加速26%
- 现代趋势: 5D并行 (TP+EP+DP+PP+CP, Context Parallelism)
- 挑战: 并行度组合空间爆炸，需要自动搜索最优配置

关键要点: 生产级MoE系统普遍采用4-5维混合并行，核心原则: NVLink内放EP/TP，跨节点放PP/DP，DualPipe实现近零通信开销。

GShard, Switch, DeepSpeed-MoE, DeepSeek-V3 各自技术报告

5.4 All-to-All通信深度解析

背景: All-to-All是MoE独有的通信模式 (稠密模型使用AllReduce)。每个GPU需将不同的数据发往不同的目标GPU, 这比AllReduce更难优化, 因为通信模式是动态的、由路由决策决定的。

All-to-All工作原理

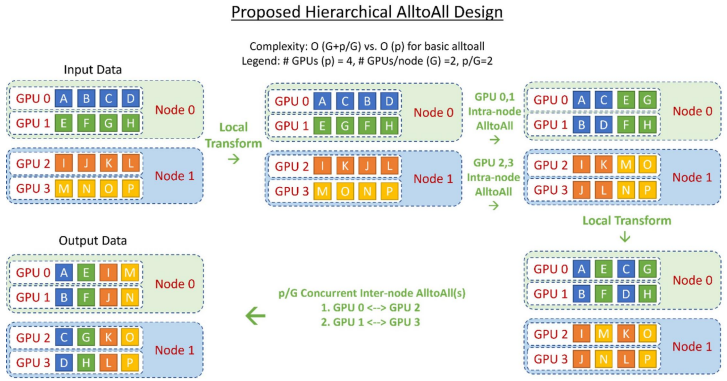
- 每个GPU向所有其他GPU发送不同的数据块
- 等价于分布式矩阵转置 (transpose/reshuffle)
- 输入: GPU_i持有token_i的路由结果
- 输出: GPU_j收到所有发给其专家的token
- vs AllReduce: 所有GPU执行相同归约操作
- vs All-to-All: 每对GPU间的数据内容和大小不同
- 动态路由 → 可变消息大小 → 需要容量缓冲区

为什么All-to-All更难优化?

- 负载不均: 热门专家接收更多token, 冷门专家闲置
- 动态模式: 每步的通信模式由路由决定, 无法静态优化
- 网络拓扑敏感: 跨节点All-to-All带宽远低于节点内
- TPU torus: O(sqrt(D))成本; GPU集群: 更高且不均匀
- 内存开销: 需为最大可能通信量预分配缓冲区
- 与计算重叠困难: 通信依赖路由结果, 难以提前发起
- NCCL优化有限: All-to-All不如AllReduce成熟

最新优化方案:

- NVIDIA Hybrid-EP: 节点内All-to-All + 跨节点选择性通信, 吞吐提升514%
- DeepSeek-V3: 将All-to-All与计算完全重叠, 通信开销近零
- FP8量化: 对All-to-All传输数据进行per-token量化, 带宽减半
- MegaScale-MoE: 将MoE层限制在单节点内 (NVLink), 避免跨节点All-to-All
- Lancet: 全图优化, 将非MoE计算与All-to-All重叠, 通信开销减少77%



关键要点: All-to-All是MoE的独特通信瓶颈 -- 动态路由导致不可预测的通信模式, Hybrid-EP和DualPipe等方案正在系统性地解决这一挑战。

NVIDIA (2024) Hybrid Expert Parallelism | DeepSeek-V3 (2024) | Lancet MLSys 2024

PART 06

MoE训练加速技术

DeepSpeed-MoE · 通信优化 · 异构硬件

6.1 DeepSpeed-MoE: 端到端MoE训练与推理框架

背景: DeepSpeed-MoE (Rajbhandari et al. 2022, ICML) 是微软提出的综合性MoE框架，覆盖训练和推理全流程。该框架首次系统性地将5种并行策略与MoE结合，并引入PR-MoE金字塔结构，在保持质量的同时大幅降低训练和推理成本。

训练效率

- 5种并行: Data + Tensor + Pipeline + Expert + ZeRO
- **训练成本: 5x计算节省 (相比等质量稠密模型)**
- 例: 350M+MoE-128达到1.3B稠密模型质量
- **吞吐提升: 5.3x (372 vs 70 samples/sec)**

PR-MoE: 反向金字塔设计

- 浅层: 16个专家，深层: 128个专家
- **效果: 模型大小减少3x，质量无损失**

推理优化

- 多GPU推理: EP + TP组合，专家分布在多卡上
- **延迟优化: 最高7.3x延迟降低**
- 万亿参数: 推理延迟控制在25ms以内

规模验证

- 最大训练规模: 4.5T参数 (3.5T MoE + 1T稠密)
- NLG任务: MoE在0-shot/few-shot上显著优于稠密模型
- 开源: 集成于DeepSpeed库，已被广泛采用

关键要点: DeepSpeed-MoE提供端到端MoE解决方案: 5x训练成本节省，7.3x推理加速，PR-MoE金字塔设计进一步减少3x模型大小。

Rajbhandari et al. (2022) DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training | ICML 2022

6.2 通信优化: 从拓扑感知到计算-通信重叠

核心挑战: All-to-All通信是MoE训练的主要瓶颈。近两年的研究从拓扑感知、计算重叠、通信压缩三个方向系统性地优化这一瓶颈, 多项工作已在生产系统中验证。

MegaScale-MoE (ByteDance, 2025)

- 核心思想: 将MoE层限制在单节点内 (NVLink通信)
- 节点内All-to-All: NVLink 900GB/s vs 跨节点IB 400Gb/s
- **加速效果: 1.65-1.88x端到端吞吐提升**

DeepSeek-V3 DualPipe

- 双向流水线: 前向和后向pass同时在管道中双向流动
- 计算-通信完全重叠: All-to-All隐藏在计算之后
- **效果: 近零All-to-All开销 (相比总训练时间)**

Lancet (MLSys 2024)

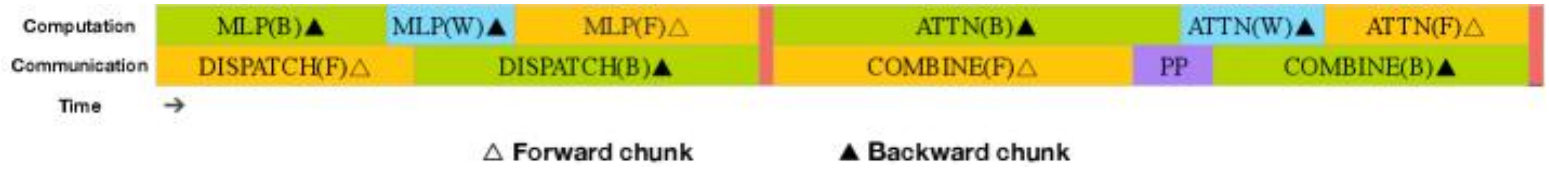
- 全图优化 (Whole-Graph Overlap): 非MoE计算与All-to-All重叠
- **通信开销减少77%: 将Attention计算时间用于隐藏通信**

COMET (2025)

- 线程块专用化 (Thread Block Specialization): 不同SM负责计算或通信
- **H800/L20加速: 1.96x吞吐提升**

FP8通信压缩

- Per-token量化: 每个token独立计算scale, 最小化精度损失
- All-to-All带宽减半: FP16→FP8, 训练质量几乎无损
- DeepSeek-V3已在生产中使用FP8 All-to-All通信



关键要点: 通信优化三板斧: 拓扑感知 (MegaScale-MoE, 1.88x), 计算重叠 (DualPipe, 近零开销; Lancet, -77%), 数据压缩 (FP8, 带宽减半)。

6.3 异构硬件加速: GPU混合与CPU卸载

背景: MoE模型的稀疏激活特性天然适合异构硬件部署: 高频使用的Attention层放在高端GPU上, 低频激活的专家FFN可以卸载到低端GPU或CPU, 从而提升整体硬件利用率并降低成本。

HeterMoE (2025): 多代GPU协同

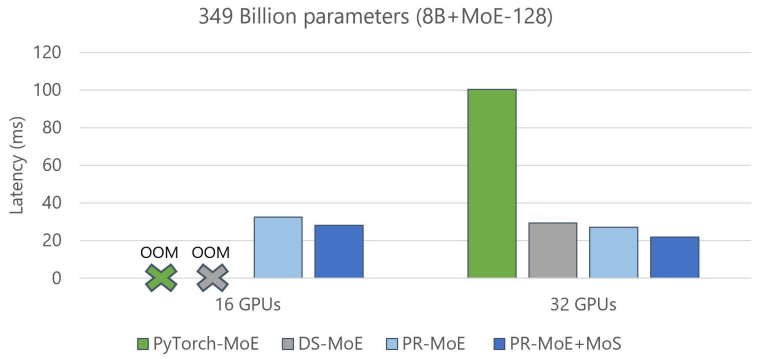
- 架构: Attention层在新GPU (如A100) + 专家FFN在旧GPU (如V100)
- 动机: 数据中心往往有多代GPU共存, 传统方案无法混用
- 调度: 基于专家负载动态分配GPU资源
- **加速效果: 最高2.3x加速 (相比仅用旧GPU)**
- 成本: 相同性能下硬件成本降低40-60%
- 挑战: 异构GPU间通信带宽不对称, 需精心调度

CPU卸载方案

- ES-MoE: 将不活跃专家卸载至CPU内存
  动态专家放置: 根据路由历史预测下一步需要的专家
- Fiddler (ICLR 2025): CPU-GPU逐层协调
  每层: 热门专家留GPU, 冷门专家CPU计算
  延迟: 单卡可运行Mixtral 8x7B, 延迟可控
- Pre-gated MoE (ISCA 2024): 预测性专家迁移
  提前一层预测路由, 重叠CPU→GPU迁移与当前层计算

异构方案对比总结:

方案	硬件组合	核心技术	加速比	适用场景
HeterMoE	新GPU+旧GPU	异构调度	2.3x	多代GPU数据中心
ES-MoE	GPU+CPU	动态卸载	1.5-2x	GPU显存不足
Fiddler	GPU+CPU	逐层协调	可变	单卡推理大模型
Pre-gated	GPU+CPU	预测迁移	1.4-1.8x	推理延迟敏感



关键要点: 异构硬件是MoE降本增效的重要方向: 多代GPU混用 (HeterMoE, 2.3x), CPU卸载 (ES-MoE/Fiddler), 预测迁移 (Pre-gated) 各有适用场景。

HeterMoE (2025) | ES-MoE | Fiddler ICLR 2025 | Pre-gated MoE ISCA 2024

本课小结: Session 2 — 路由机制与训练优化

路由算法 (5类)

- Token Choice: Top-1/Top-2路由
- Expert Choice: 专家主动选token
- Loss-Free Balancing: 无辅助损失
- Hash/Random: 确定性路由
- Soft MoE: 连续可微路由

负载均衡 (3方法)

- 辅助损失: l_aux / l_importance
- 容量因子: CF控制缓冲大小
- Loss-Free: 偏置项动态调节

训练稳定性

- Router Z-Loss正则化
- Jitter Noise随机扰动
- BFloat16 / 选择性精度

细粒度专家

- 知识冗余与混杂问题
- 细粒度分割: mN专家mK激活
- 共享专家隔离: 通用知识分离

分布式训练

- EP: 专家并行, All-to-All通信
- 4维并行: DP+TP+PP+EP
- DualPipe双向流水线

加速技术

- DeepSpeed-MoE: 5维并行框架
- 通信优化: 拓扑+重叠+压缩
- 异构硬件: GPU混合+CPU卸载

关键要点: Session 3预告: 推理部署 (Inference Deployment)、前沿模型分析 (Frontier Models)、模型压缩 (Compression)、未来方向 (Future Directions)。

第三课时

前沿应用、推理部署与未来方向

Algorithm → Training → Now: Inference & Deployment

回顾: 前两节核心串联

背景: 第一节我们学习了MoE的基础算法原理, 第二节深入了训练系统设计。本节将聚焦推理部署(Inference & Deployment)与前沿应用, 完成从"理论→训练→推理"的完整链条。

第一节: 算法基础 (Algorithm)

- MoE核心公式: $y = \Sigma_i G(x)_i \cdot E_i(x)$

五种路由算法: Token Choice (Top-K), Expert Choice, Hash Routing, BASE Layer, DeepSeek Auxiliary-Free

三种负载均衡方法:

1. Auxiliary Loss (Switch Transformer)
2. Expert Choice (反转路由)
3. Auxiliary-Loss-Free (DeepSeek偏置项)

专家专业化: 不同专家自发学习不同语义模式

第二节: 训练系统 (Training)

- Expert Parallelism (EP): 专家分布到不同设备

All-to-All通信: Dispatch→Compute→Combine
混合并行: EP + TP + PP + DP 组合策略
容量因子 (Capacity Factor): 控制缓冲区大小
训练稳定性: 路由崩塌、Z-Loss、Router Z-Init
DeepSeek DualPipe: 计算与通信重叠
FP8混精度训练: 从训练伊始使用低精度

第三节: 推理部署与前沿应用 (Inference & Deployment)

- PART 01: 前沿MoE大模型 — DeepSeek-V3, Mixtral, Qwen3, Llama 4, GPT系列, Gemini
PART 02: MoE推理系统设计 — 推理 vs 训练的差异, EP for Inference, Prefill-Decode分离
PART 03: 模型压缩与高效部署 — 剪枝、蒸馏、量化、Expert Offloading
PART 04: 未来方向与开放问题 — MoE + MLA, Multimodal MoE, 边缘部署

核心问题: 如何将数千亿参数的MoE模型高效部署到生产环境?

训练与推理面临完全不同的系统挑战 — 推理对延迟敏感, 批大小小, 负载更不均衡

本节主线: 从"训练好的MoE模型"到"高效服务用户请求" — 推理系统设计是MoE产业化的关键瓶颈。

PART 01

前沿MoE大模型

GPT · Gemini · DeepSeek · Mixtral · Llama · Qwen

前沿MoE大模型: 架构全景对比 (2024-2025)

背景: 截至2025年, 几乎所有前沿大模型都采用了MoE架构。以下数据均来自各模型官方论文或技术报告, 标注了具体来源。未经官方确认的信息明确标出。

模型	总参数	激活参数	专家数	路由	注意力	来源
DeepSeek-V3	671B	37B	256+1共享	Top-8+1	MLA	arXiv:2412.19437
DeepSeek-R1	671B	37B	256+1共享	Top-8+1	MLA	同架构, RL训练
Mixtral 8x7B	46.7B	12.9B	8	Top-2	GQA	arXiv:2401.04088
Qwen3	235B	22B	128	Top-8	GQA	arXiv:2505.09388
Llama 4 Scout	109B	17B	16	Top-1	GQA	Meta官方博客
Llama 4 Maverick	~400B	17B	128	Top-1	GQA	Meta官方博客
GPT-OSS-120B	117B	5.1B	128	Top-4	-	OpenAI开源
Gemini 3 Pro	未公开	未公开	稀疏MoE	未公开	-	官方确认MoE
GPT-4 / GPT-5	未公开	未公开	未确认	未确认	-	业界推测, 未官方确认

关键趋势观察

- 专家数量持续增加: 8 (Mixtral) → 16 (Scout) → 128 (Qwen3/Maverick/GPT-OSS) → 256 (DeepSeek) — 更细粒度的专家分工
  激活比例持续降低: 27.6% (Mixtral) → 9.4% (Qwen3) → 5.5% (DeepSeek-V3) → 4.3% (GPT-OSS) — 更极致的稀疏激活
  共享专家成为标配: DeepSeek (+1 shared), Llama 4 (+1 shared) — 共享专家捕获通用知识, 路由专家学习特化知识
  注意力机制创新: MLA (DeepSeek) 大幅降低KV Cache, 与MoE形成互补 — 两者都是"用结构换效率"的设计哲学
  开源与闭源并行: DeepSeek/Qwen/Llama/GPT-OSS完全开源, 而GPT-4/5/Gemini保持闭源 — MoE架构选择已无秘密

2025年共识: MoE已成为前沿大模型的标准架构, 核心分歧在于专家粒度(8 vs 256)、路由策略(Top-1 vs Top-8)和是否使用共享专家。

数据来源: 各模型官方论文与技术报告, 2024-2025; GPT-4/5信息为业界推测

DeepSeek-V3: 架构深度解析

背景: DeepSeek-V3是目前公开文档最详尽的前沿MoE模型。它在架构、训练和系统三方面均有重要创新，被广泛认为是2024年最具影响力的开源大模型之一。以下分析基于其技术报告(arXiv:2412.19437)。

架构核心创新

1. MLA (Multi-head Latent Attention)

将KV Cache压缩到低维潜空间
KV Cache从 $2 \times n_h \times d_h$ 降至 $d_c + d_r$
推理时KV Cache减少约93.3% (vs MHA)
性能与MHA持平，推理吞吐大幅提升

2. DeepSeekMoE: 细粒度256+1专家

256个路由专家 + 1个共享专家 (Shared Expert)
每token激活8个路由专家 + 1共享专家
共享专家: 捕获所有token共需的通用知识
路由专家: 学习细粒度的领域特化知识

3. Auxiliary-Loss-Free负载均衡

传统辅助损失会干扰主训练目标
引入偏置项b_i，动态调节路由概率
γ=0.001: 过载专家降低偏置，空闲专家提升
结果: 负载均衡与模型质量完全解耦

4. Multi-Token Prediction (MTP)

训练时同时预测下一个和后续token
推理时可用于Speculative Decoding加速

训练规模与效率

- 总参数: 671B，激活参数: 37B (5.5%)
  训练数据: 14.8T tokens
  训练精度: 从头使用FP8 mixed precision
  训练硬件: 2048 NVIDIA H800 GPUs
  训练时间: ~2个月 (vs 估计Dense同级需12个月)
  训练成本: ~$5.5M (极低于同级闭源模型)
  系统创新: DualPipe (计算通信重叠)，EP64+DP4

性能表现 (Benchmark Highlights)

- MMLU: 87.1% (vs GPT-4o 87.2%)
  MATH-500: 90.2% (超越多数闭源模型)
  Codeforces: 51.6 percentile (开源最强)
  SWE-bench Verified: 42.0% (开源SOTA)
  以1/10的训练成本达到GPT-4o同级水平
  验证了"MoE + 系统优化"的路线可行性
  R1进一步通过RL训练增强推理能力

DeepSeek-V3启示: MoE + MLA + FP8 + 系统优化 = 以$5.5M成本训练出GPT-4o级别模型，证明开源MoE路线的可行性与经济性。

Source: DeepSeek-V3 Technical Report, arXiv:2412.19437, December 2024

跨模型设计选择对比: 为什么不同?

核心问题: 前沿MoE模型在专家数量、路由策略、注意力机制等方面做出了截然不同的选择。这些选择背后有什么权衡? 没有"最优"解, 只有针对不同约束的"最适"解。

设计维度	选项A	选项B	权衡分析
路由策略	Top-1/Top-2 (Mixtral, Llama4)	Top-8 (DeepSeek, Qwen3)	Top-K小→推理快但路由压力大; Top-K大→更稳定但通信开销高
专家粒度	8个大专家 (Mixtral: 每个7B)	256个小专家 (DeepSeek: 每个~2B)	大专家→EP简单, 负载不均风险高; 小专家→分工精细, 通信/调度复杂
共享专家	有 (DeepSeek, Llama4)	无 (Mixtral)	共享专家捕获通用模式, 减轻路由专家负担; 但增加每token必须计算量
注意力	标准MHA/GQA (Mixtral, Llama4)	MLA (DeepSeek)	MLA大幅降低KV Cache, 推理更高效; 但训练复杂度更高, 与MoE互补增益显著
训练方法	标准预训练 (Mixtral, Qwen3)	预训练 + RL (DeepSeek-R1)	RL增强推理能力(Chain-of-Thought); 但RL训练不稳定, 需要reward模型
负载均衡	辅助损失 (Mixtral, Switch)	无损负载均衡 (DeepSeek)	辅助损失简单但干扰训练目标; 无损方法更优但实现更复杂

设计哲学总结: Mixtral = "简洁有效" (8个大专家, Top-2, 无共享专家); DeepSeek = "极致优化" (256细粒度专家, MLA, 无损均衡); Llama 4 = "折中方案" (16-128专家, Top-1, 有共享); Qwen3 = "跟随验证" (128专家, Top-8, 验证DeepSeek路线)。

核心洞察: 专家粒度×路由Top-K×共享专家 三个维度的组合, 决定了MoE模型的效率-质量-部署难度权衡。没有统一最优解。

分析基于各模型官方论文, 2024-2025

PART 02

MoE推理系统设计

从训练到部署：不同的系统挑战

推理 vs 训练: 根本不同的系统需求

核心观点: MoE推理与训练面临根本不同的系统挑战。训练追求吞吐(Throughput), 推理追求延迟(Latency); 训练有大批量平摊通信, 推理每次仅处理少量token。这些差异导致训练时的并行策略不能直接复用到推理。

训练阶段 (Training)

- **目标: 最大化吞吐量 (tokens/second)**
  批大小: 数千~数万个token → All-to-All效率高
  内存: 需存储梯度 + 优化器状态 (3×模型大小)
  通信: AllReduce (梯度同步) + All-to-All (EP)
  负载均衡: 大批量统计平滑, 专家负载较均匀
  计算: 前向 + 反向 + 优化器更新
  硬件: GPU利用率高 (计算密集 compute-bound)
  延迟不敏感: 每步耗时可以较长, 关键是总训练时间

推理阶段 (Inference)

- **目标: 最小化延迟 (ms/token) + 最大化吞吐**
  批大小: 单请求1个token (decode) → All-to-All浪费
  内存: 只需模型权重 + KV Cache (无梯度/优化器)
  通信: 仅All-to-All (EP), 无梯度同步
  负载均衡: 小批量统计波动大, 专家负载极不均
  计算: 仅前向传播
  硬件: GPU利用率低 (内存带宽瓶颈 memory-bound)
  延迟极敏感: 用户等待每个token, TTFT是核心指标

Prefill-Decode分离 (Disaggregated Serving)

- Prefill阶段 (首次处理所有输入token): 计算密集型(Compute-bound), 类似训练 → 需要高算力GPU, 批处理效率高
  Decode阶段 (逐token生成输出): 内存带宽瓶颈型(Memory-BW-bound), 每步仅1 token → 需要高带宽, 低延迟
  分离策略: Prefill tier (高算力硬件, 如H100 SXM) + Decode tier (高带宽硬件, 或更多GPU分摊KV Cache)
  优势: 各阶段独立扩缩容, 硬件利用率提升 — DeepSeek生产系统采用此策略
  MoE特殊性: Prefill时All-to-All通信量大(多token), Decode时All-to-All消息极小(单token) — 需要不同的EP策略

关键区别: 训练是Compute-bound + 大批量; 推理是Memory-BW-bound + 小批量 → MoE推理需要全新的并行策略和负载均衡方案。

参考: DeepSeek-V3 Tech Report; Zhong et al. DistServe, OSDI 2024

推理阶段的Expert Parallelism与负载均衡

核心挑战: MoE推理中EP面临比训练更严重的负载不均问题。Decode阶段每次仅处理少量token，路由的随机性无法被大批量平滑，导致某些专家过载而其他专家空闲。DeepSeek报告显示，无EPLB时工作负载不均衡可达2倍以上。

推理EP核心问题

- Decode阶段: 每步仅1 token/request生成
  All-to-All消息极小 → 通信延迟占比极高
  负载不均衡放大: 热门专家等待，冷门专家空闲
  DeepSeek实测: 最忙/最闲专家负载比 > 2.0×
  推理GPU利用率 < 30% (无优化时)
  解决目标: 让每个GPU的专家计算量尽量均等
  约束: 不能修改模型权重，只能调整专家放置

EPLB (Expert Parallel Load Balancing)

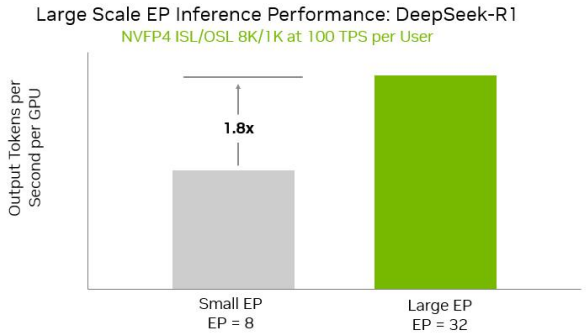
- **策略1: 冗余专家 (Redundant Experts)**
  对热门专家复制到多个GPU上
  热门专家处理的token被分摊到多个副本

  策略2: 层级打包 (Hierarchical Packing)
  节点内: 考虑NVLink连接，优先本地通信
  节点间: 全局最优分配，最小化跨节点流量

  策略3: 全局打包 (Global Packing)
  基于历史负载数据，全局优化专家到GPU映射
  将负载相似的专家分到不同GPU，互相平衡

DeepSeek生产环境推理性能

指标	数值	说明
输入吞吐	73.7k tok/s	每节点，Prefill
输出吞吐	14.8k tok/s	每节点，Decode
EP配置	EP64+DP4	Prefill/Decode不同EP
EPLB增益	+30%吞吐	冗余专家+层级打包



EPLB核心思想: 通过冗余专家复制 + 层级/全局打包，消除推理时的负载不均衡，吞吐提升约30% — 这是MoE推理产业化的关键技术。

Source: DeepSeek-V3 Tech Report, Section 5; DeepSeek EPLB开源工具

NVIDIA NVL72: 专为MoE推理设计的硬件

背景: NVIDIA GB200 NVL72是首个明确为MoE推理优化的硬件平台。72颗GPU通过NVSwitch全互联, 提供130 TB/s总NVLink带宽, 使得Wide-EP (每个GPU仅放少量专家) 成为可能, 而不受通信瓶颈制约。

GB200 NVL72 硬件架构

- 72颗GB200 GPU, 通过NVSwitch全互联
  130 TB/s NVLink总带宽 (每GPU 1.8 TB/s)
  关键: 非直连拓扑, 所有通信经NVSwitch
  → 任意GPU对之间带宽相等 (flat topology)
  → 与传统NVLink直连 (8 GPU mesh) 根本不同
  每GPU: 192GB HBM3e, ~8 TB/s内存带宽
  总系统: 13.8TB GPU内存 → DeepSeek-V3全模型
  支持EP72: 每GPU放 256/72 ≈ 3-4 个专家

NVL72上的EPLB策略

- **Static EPLB (静态负载均衡)**
  基于历史路由数据预计算专家→GPU映射
  热门专家复制到多个GPU (冗余副本)
  映射表在服务启动时加载, 运行中不变

  Dynamic EPLB (动态负载均衡)
  运行时监控各专家实际负载
  周期性重新分配专家到GPU
  利用NVSwitch: 迁移专家权重仅需毫秒级

  NVSwitch的关键优势:
  所有GPU等距 → 无"远程"通信惩罚
  专家迁移不影响其他GPU的通信

NVL72 vs 传统集群对比

特性	传统8-GPU集群	GB200 NVL72
GPU拓扑	NVLink+InfiniBand	**72 GPU全NVSwitch**
EP配置	EP8, 跨节点慢	**EP72, 无瓶颈**
A2A延迟	高 (~5μs+)	低 (~1μs)
动态EPLB	困难	容易 (等距)

NVL72的设计哲学: 通过NVSwitch将72 GPU变成"一个超级GPU", 使MoE推理的All-to-All通信不再是瓶颈 — 硬件与算法协同设计(co-design)。

Source: NVIDIA GB200 NVL72技术白皮书; DeepSeek EPLB论文

Meta N-D Parallelism: 多维并行推理系统

背景: Meta在2025年发布了其MoE推理系统设计, 采用N维并行(N-D Parallelism)策略, 将CP(Context Parallel)、PP(Pipeline Parallel)、EP(Expert Parallel)、TP(Tensor Parallel)组合使用, 并引入分离式服务(Disaggregated Serving)。

N-D Parallelism 多维并行

每个维度解决不同问题:

CP (Context Parallelism): 长序列切分到多GPU
PP (Pipeline Parallelism): 模型按层切分, 流水线执行
EP (Expert Parallelism): 专家分布到不同GPU (MoE核心)
TP (Tensor Parallelism): 单层权重切分到多GPU
DP (Data Parallelism): 多副本处理不同请求

分离式服务 (Disaggregated Serving)

- **Prefill Tier (预填充层)**
  高算力硬件 (如H100 SXM)
  批量处理输入, 计算密集型
  CP + EP组合, 处理长上下文

  Decode Tier (解码层)
  高内存带宽硬件
  逐token生成, 内存带宽瓶颈
  EP + TP组合, 最小化延迟

异构硬件趋势 (Heterogeneous HW)

- **MoE推理正在推动异构硬件部署:**
  Prefill: 高算力GPU (H100/GB200)
  Decode: 高带宽但较低算力的加速器
  Expert存储: 可扩展到CPU内存/NVMe
  调度器: 根据请求特征分配到合适硬件
  Meta Llama 4部署可能采用类似架构
  趋势: 从"同构集群"到"异构计算池"

Meta的N-D Parallelism启示: MoE推理需要多维并行组合 + Prefill/Decode分离 + 异构硬件, 系统复杂度远超Dense模型推理。

Source: Meta Llama 4 Technical Blog; Meta Infrastructure Talks, 2025

PART 03

MoE模型压缩与高效部署

剪枝 · 蒸馏 · 量化 · Expert Offloading

MoE模型剪枝 (Pruning): 去除冗余专家与参数

背景: MoE模型参数量巨大(如DeepSeek-V3 671B), 部署成本极高。剪枝(Pruning)通过移除不重要的专家或专家内部的冗余参数来减小模型大小, 同时尽量保持性能。MoE的稀疏激活特性使其天然适合剪枝 – 许多专家在特定任务中很少被激活。

四种MoE剪枝方法

1. MoE-Pruner: 路由感知一次性剪枝
利用Router权重评估专家重要性
直接移除不重要的整个专家, 无需再训练
优势: 部署即剪枝, 极快速; 限制: 精度损失较大

2. SlimMoE: 专家内部神经元剪枝 + 蒸馏
不移除整个专家, 而是剪掉专家内部冗余神经元
多阶段蒸馏: 先剪枝, 再用原模型蒸馏恢复精度
更精细的压缩, 性能保持更好

3. MoE-I²: 双层级剪枝 (Inter + Intra)
Inter-Expert: 移除25%不重要的专家
Intra-Expert: 对剩余专家做低秩分解 (再压37.5%)
两级叠加: 总压缩率 ~53%

4. MoNE: 轻量"新手"网络替换
识别冗余专家 → 用小型"novice"网络替换
新手网络: 参数仅为原专家的1/4~1/8
保留专家多样性, 同时大幅减少参数量

关键实验结果

方法	模型	压缩率	性能保留
MoE-Pruner	Mixtral 8x7B	50%专家	~95%
SlimMoE	Mixtral 8x7B	50%稀疏	~99%
MoE-I²	Mixtral 8x7B	~53%	~97%
MoNE	通用MoE	~60%	~96%

剪枝策略选择指南

- 需要快速部署, 无再训练资源 → MoE-Pruner
  追求最高精度保留 → SlimMoE (需蒸馏成本)
  需要极致压缩率 → MoE-I² (双层级叠加)
  保留专家多样性 → MoNE (轻量替换)
  实际部署: 通常剪枝+量化组合使用
  先剪枝减少专家数, 再量化降低每参数位宽
  可将Mixtral从47B降至~10B等效大小

MoE剪枝核心: 利用稀疏激活特性, 50%压缩可保留95-99%性能。SlimMoE(蒸馏辅助)效果最佳, MoE-I²(双层级)压缩率最高。

Sources: MoE-Pruner 2024; SlimMoE 2024; MoE-I² 2024; MoNE 2024

知识蒸馏、量化与Expert Offloading

背景: 除剪枝外, 蒸馏(Distillation)、量化(Quantization)和Expert Offloading是MoE模型压缩部署的三大互补技术。蒸馏将MoE知识转移到更小模型; 量化降低每参数位宽; Offloading利用CPU/NVMe扩展可用内存。

知识蒸馏 (Knowledge Distillation)

- **MoE→Dense蒸馏: 将大型MoE压缩为小型Dense**
  Teacher (MoE 671B) → Student (Dense 7B/14B)
  DeepSeek-R1蒸馏系列: 1.5B到70B多种大小
  Expert-wise蒸馏: 逐专家对齐知识表征

  Shadow-MoE: 路由模式蒸馏
  不仅蒸馏输入→输出映射
  还蒸馏路由决策模式(哪些专家处理哪些token)
  结果: Student学到更好的"分工"策略
  蒸馏后Dense模型可达MoE 80-90%性能

量化 (Quantization)

- **混合精度量化: 关键层FP16, 专家层INT4/INT8**
  Router权重保持高精度(路由决策敏感)
  专家FFN权重可大幅量化(冗余度高)

  MxMoE: MoE专用混合精度框架
  根据专家重要性自适应分配量化位宽
  热门专家保持较高精度, 冷门专家激进量化

  FP8统一训练-推理: DeepSeek-V3从训练起即使用FP8
  训推一致, 无量化精度损失
  QuantMoE-Bench: MoE量化评估基准

Expert Offloading: GPU-CPU混合部署

- 核心思想: MoE的稀疏激活特性意味着大部分专家在任意时刻都不被使用 → 可以将不活跃专家放在CPU/NVMe, 仅将被激活专家加载到GPU
  Fiddler (ICLR 2025): 逐层判断最优执行位置 — 如果专家在GPU上, 直接计算; 否则将输入发到CPU计算
  关键决策: 哪些专家常驻GPU (热门), 哪些放CPU (冷门) — 基于历史路由频率动态调整
  带宽挑战: PCIe 5.0 ~64 GB/s vs NVLink 900 GB/s → CPU↔GPU传输是瓶颈, 需要预取(Prefetch)和缓存(Cache)策略
  适用场景: 内存受限设备 (单GPU部署), 边缘计算, 成本敏感的推理服务
  与剪枝/量化互补: 先量化减少单专家大小, 再Offloading管理GPU-CPU放置 → 单张4090可运行Mixtral

MoE压缩三件套: 蒸馏(MoE→Dense, 保留80-90%能力) + 量化(FP8/INT4, 2-4×压缩) + Offloading(GPU-CPU混合, 突破显存限制) — 组合使用效果最佳。

Sources: Shadow-MoE 2024; MxMoE 2024; Fiddler (ICLR 2025); QuantMoE-Bench 2024

PART 04

MoE在视觉与多模态

V-MoE · MoE-LLaVA · MoME

4.1 V-MoE: 视觉领域的MoE (Vision MoE)

背景: MoE不仅适用于语言模型 -- Google将稀疏MoE成功应用于Vision Transformer (ViT), 证明了MoE的计算效率优势可以跨模态迁移。V-MoE以更少的推理计算量达到了更高的图像分类精度。

V-MoE核心设计 (Google, 2022)

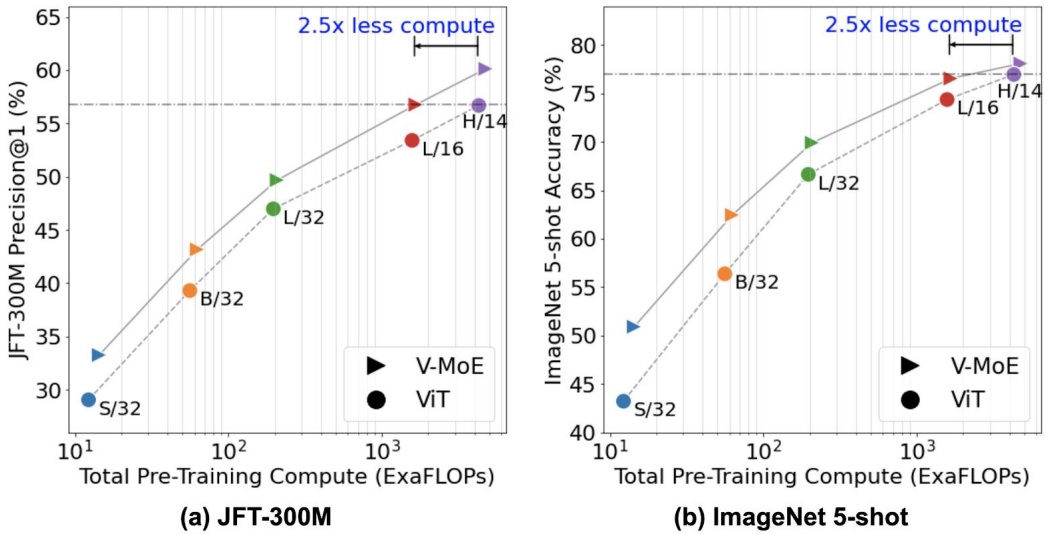
- 15B参数的Vision Transformer, 采用稀疏MoE架构
- 将ViT中部分Dense FFN层替换为稀疏MoE层 (与语言模型相同思路)
- 每个image patch token经过路由选择专家处理
- Top-K路由机制与语言MoE一致, 但针对视觉特征做了适配

关键实验结果

- **ImageNet Top-1准确率: 90.35% (刷新当时记录)**
- **推理FLOPs仅为同等Dense ViT的约一半**
- 训练效率: 相同计算预算下, MoE版本收敛更快
- Routing分析: 不同专家自动学习关注不同视觉特征区域

核心启示

- MoE的"稀疏激活 = 计算效率"范式不局限于NLP
- 视觉token的路由模式与语言token存在显著差异, 值得深入研究
- 为后续多模态MoE (如MoE-LLaVA) 奠定了基础



关键要点: V-MoE证明MoE的计算效率优势可跨模态迁移: 15B参数ViT达到90.35% ImageNet准确率, 推理FLOPs减半。

Riquelme et al. (2021) "Scaling Vision with Sparse Mixture of Experts" | Google Research

4.2 MoE在多模态大模型中的应用

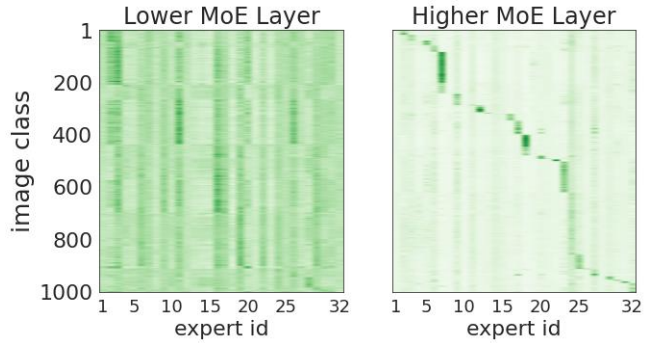
背景: 多模态大模型需要同时处理文本、图像、视频等不同模态的信息。MoE的稀疏路由天然适合多模态场景: 不同模态的token可以被路由到不同的专家, 实现模态感知的专家分工。

MoE-LLaVA (2024年1月)

- **仅3B活跃参数, 性能可比LLaVA-1.5-7B**
- **在幻觉 (Hallucination) 基准上超越LLaVA-1.5-13B**
- 通过MoE实现"小参数激活, 大参数容量"的平衡
- 证明MoE在多模态场景下的参数效率优势

MoME (NeurIPS 2024)

- MoE与FFN并行结构 (非替换, 而是增强)
- 模态感知路由 (Modality-Aware Routing): 根据输入模态调整路由策略
- 不同模态token天然需要不同的专家处理模式



更多前沿工作

- Shared Expert增强: 保留部分共享专家处理跨模态公共知识
- MoCHA (2025): Connector层MoE, 在视觉-语言连接器中引入MoE
- **Gemini: 从训练开始就采用稀疏MoE, 原生多模态训练**

多模态路由的核心挑战

- 不同模态的token分布差异大 (图像patch vs 文本token)
- 负载均衡更困难: 图像token数量远多于文本token
- 专家是否应该按模态特化? 还是学习跨模态表示?
- 路由策略需要同时考虑模态属性和语义内容

关键要点: MoE天然适合多模态: 不同模态token可路由至不同专家。MoE-LLaVA以3B活跃参数超越13B Dense模型, Gemini原生MoE多模态。

MoE-LLaVA (Lin et al. 2024) | MoME (NeurIPS 2024) | Gemini (Google 2024)

PART 05

开放问题与未来方向

路由可解释性 · 专家利用率 · 硬件协同设计

5.1 路由可解释性与专家特化 (Routing Interpretability)

背景: MoE模型中, 路由器(Router)决定哪些token送给哪些专家。一个关键问题是: 专家到底学到了什么? 路由决策是否可解释? 理解这些有助于改进路由算法和诊断训练问题。

专家特化分析 (Expert Specialization)

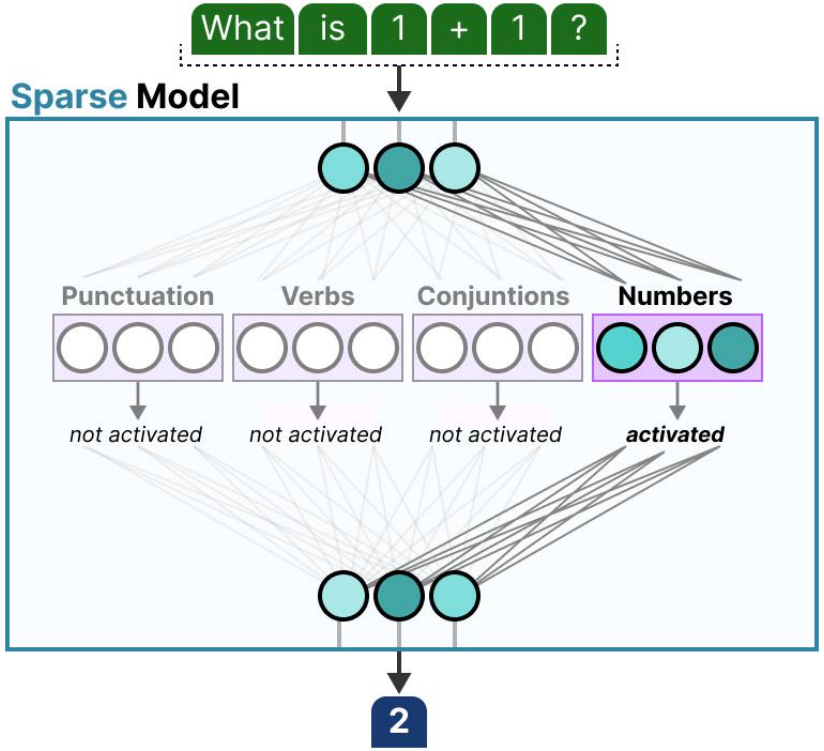
- 理论假设: 不同专家应该学习处理不同类型的知识 (语法、语义、领域等)
- **Mixtral实证: 专家并非按领域特化, 更多按句法模式 (Token级路由)**
- 某些专家偏好处理标点、连接词; 另一些偏好名词短语、技术术语

Token级 vs 序列级路由的影响

- Token级路由 (Mixtral等): 专家按token类型特化, 非按主题特化
- 序列级路由: 专家按领域/任务特化的可能性更高
- 不同粒度的路由产生完全不同的专家特化模式

OpenMoE的重要发现: "Context-Independent" 路由

- **路由决策主要由Token ID决定, 而非上下文语义**
- 同一个token (如"the") 无论出现在什么上下文中, 大概率被路由到相同专家
- 这意味着当前路由器可能没有充分利用上下文信息
- 可视化方法: 路由概率热力图 (Routing Probability Heatmap) 跨层分析专家选择模式



关键要点: 当前MoE路由主要依赖Token ID而非上下文 (Context-Independent), 专家更多按句法而非语义特化。提升路由的上下文感知能力是重要方向。

Mixtral routing analysis | OpenMoE (Xue et al. 2024) | Routing probability heatmap visualization

5.2 专家利用率优化 (Expert Utilization)

背景: MoE模型的一个核心局限: 每次前向传播只激活Top-K个专家, 其余专家的参数完全闲置。这意味着大量显存被未使用的参数占据。如何提高专家利用率, 或在不损失性能的前提下减少闲置专家, 是MoE系统优化的关键方向。

核心问题

- **Top-K路由: 每次前向传播仅激活K个专家, 其余N-K个完全闲置**
- 例: DeepSeek-V3有256个专家, 每token仅激活8个, 利用率3.1%
- 闲置专家占据显存但不贡献计算 -- 资源浪费

优化策略一: 专家合并 (Expert Merging)

- 将功能相似的专家合并, 减少总专家数但保持性能
- 基于专家权重相似度或路由相关性进行聚类合并
- 合并后模型更小, 更易部署, 但需权衡精度损失

优化策略二: 自适应K与动态专家数

- 自适应K (Adaptive K): 简单token用少量专家, 复杂token用更多专家
- u-MoE: 测试时剪枝, 作为micro-grained MoE实现细粒度激活
- **DYNMOE (ICLR 2025): 训练期间自动添加/删除专家, 动态调整专家总数**
- 每层的最优专家数可能不同 -- 浅层需要少量专家, 深层需要更多

关键要点: 提升专家利用率的三条路: 合并冗余专家减少浪费, 自适应K动态调整激活数, DYNMOE训练时自动调整专家总数。

u-MoE (2024) | DYNMOE (ICLR 2025) | Expert merging for MoE compression

5.3 MoE与推理能力 (MoE for Reasoning)

背景: 随着Chain-of-Thought (CoT) 推理和强化学习在LLM中的成功, 一个自然的问题浮现: MoE架构能否帮助提升逐步推理能力? 不同推理步骤是否需要不同的专家组合?

DeepSeek-R1: MoE + RL推理的突破

- 基于DeepSeek-V3的MoE架构, 通过强化学习训练推理能力
- **在数学推理、代码生成等任务上取得突破性成果**
- 证明MoE架构与RL推理训练兼容且高效
- 稀疏激活使得长链推理的计算成本可控

推理任务的最优稀疏性 (arXiv:2508.18672)

- 推理任务可能需要与语言建模不同的稀疏度
- 过高稀疏度可能损害推理链的连贯性
- 最优K值可能随推理步骤深度变化

开放问题: 路由是否随推理步骤自适应?

- 假设: 推理的不同阶段 (理解题意/制定计划/执行计算/验证) 需要不同专家
- 实证分析: 路由模式是否在推理链中展现阶段性变化?
- **MoE + Chain-of-Thought的交叉研究正处于早期阶段**

关键要点: DeepSeek-R1证明MoE+RL可实现高效推理。推理任务的最优稀疏度、路由自适应性是MoE+CoT交叉领域的活跃研究方向。

DeepSeek-R1 (2025) | Optimal sparsity for reasoning (arXiv:2508.18672)

5.4 硬件-软件协同设计 (Hardware-Software Co-Design)

背景: MoE模型的All-to-All通信模式对网络带宽提出了极高要求。传统硬件架构并非为这种"每对设备都需要交换数据"的通信模式设计。新一代互连技术和芯片架构正在为MoE量身定制。

互连技术演进

- **NVLink演进: GB200 NVL72提供900 GB/s带宽**
- 专为All-to-All通信模式优化 (MoE的核心通信模式)
- CXL互连: 内存扩展 + 专家卸载 (Expert Offloading)
- CXL允许GPU共享远程内存, 存放不活跃专家

芯片级创新

- Mozart系统: 树状互连 (Tree Interconnect) 用于Chiplet架构MoE
- **A3D-MoE: 3D异构集成, 延迟降低1.8-2x**
- 将专家分布在不同Chiplet上, 片上互连替代片间通信

未来硬件趋势

- **Chiplet架构为稀疏MoE工作负载量身定制**
- 专家级别的数据流优化: 减少不必要的数据搬移
- 异构计算: 不同专家可运行在不同类型的计算单元上
- 近存计算 (Processing-in-Memory) 减少专家参数加载延迟

软硬件协同设计的关键洞察

- 算法决定通信模式, 硬件决定通信成本
- 最优MoE设计需要同时考虑算法和硬件约束
- 专家数量、路由Top-K、并行策略都应与硬件拓扑匹配

关键要点: NVLink 900GB/s、CXL内存扩展、3D Chiplet架构正在为MoE的All-to-All通信量身定制。硬件与算法的协同设计是高效MoE部署的关键。

NVIDIA GB200 NVL72 | CXL Consortium | Mozart (ISCA 2024) | A3D-MoE (2024)

5.5 自适应与动态MoE (Adaptive & Dynamic MoE)

背景: 传统MoE的专家数量和路由Top-K在训练前固定，推理时不变。但不同样本、不同任务、不同层可能需要不同数量的专家。将专家数量和激活数量从静态设计变为动态运行时变量是MoE的重要演进方向。

DYNMOE (ICLR 2025)

- **训练期间自动调整每层的专家数量 (Auto-tune)**
- 基于梯度信号判断某层是否需要更多/更少专家
- 打破"所有层相同专家数"的传统设计约束
- 浅层可能只需4个专家，深层可能需要64个专家

动态路由: 可变激活专家数

- **核心思想: 难任务/难token → 激活更多专家，简单token → 少量专家**
- 实现计算量与任务难度的自适应匹配
- 推理时可根据延迟预算动态调整K值

Expert Threshold (ET) 路由 (2026)

- 为每个专家维护EMA阈值 (Exponential Moving Average)
- **门控值超过阈值的专家被激活，无需固定Top-K**
- 完全无辅助损失 (No Auxiliary Loss)，避免梯度干扰
- **未来方向: 将专家数量视为运行时自适应变量，而非静态设计选择**

关键要点: MoE正从"静态设计"走向"动态自适应": DYNMOE自动调整专家数，ET路由无需固定K和辅助损失，专家数量成为运行时变量。

DYNMOE (ICLR 2025) | Expert Threshold routing (2026) | Adaptive computation

PART 06

总结与展望

三课时知识全景图

6.1 三课时知识全景图 (Three-Session Knowledge Map)

背景: 三次课程覆盖了MoE从基础理论到前沿应用的完整知识体系。本页将三课时的核心知识点串联，展示它们之间的逻辑关系：算法决策影响系统实现，系统设计制约部署方案。

Session 1: 基础与起源

- Scaling Laws → MoE概念
- MoE历史 (1991-2017)
- 核心组件: Router + Experts
- Shazeer 2017 → GShard → Switch

Session 2: 算法与系统

- 路由算法 (Token/Expert Choice)
- 负载均衡与训练稳定性
- DeepSeekMoE架构创新
- 系统设计与加速 (EP/TP/PP)

Session 3: 前沿与未来

- 前沿模型 (DeepSeek-V3/Qwen)
- 推理优化与压缩
- 视觉/多模态MoE
- 开放问题与未来方向

知识链路: 概念基础 → 算法设计 → 系统实现 → 部署优化 → 前沿探索

核心逻辑关系

- 算法决策 → 系统影响: 路由Top-K决定All-to-All通信量，专家数决定EP并行度
- 系统设计 → 部署约束: EP/TP划分决定显存分配，通信延迟决定推理吞吐
- 部署需求 → 算法改进: 推理成本高 → 量化/剪枝/专家卸载 → 新路由算法
- 硬件演进 → 架构创新: NVLink/CXL带宽提升 → 更大规模MoE变得可行
- 理论分析 → 实践指导: Scaling Laws预测最优专家数与模型规模的关系

关键要点: MoE是算法、系统、硬件深度耦合的技术: 算法决策直接影响系统开销，硬件约束反过来指导算法改进。三课时覆盖完整闭环。

6.2 MoE系统设计关键决策树 (Decision Tree)

背景: 设计一个MoE系统需要做出一系列相互关联的决策。每个选择都有trade-off, 没有绝对最优解。以下决策树帮助系统性地思考MoE设计空间。

决策1: 专家数量与粒度

- 少量大专家 (Mixtral: 8个): 路由简单, 负载均衡容易, 但专家特化有限
- 大量小专家 (DeepSeek: 256个): 更细粒度特化, 但路由和通信更复杂

决策2: 路由Top-K选择

- Top-1 (Switch): 最低通信量, 最大稀疏度, 但鲁棒性较弱
- Top-2 (Mixtral): 平衡性能与效率, 工业界最常见选择
- Top-8 (DeepSeek-V3): 更高激活率, 更好性能, 但通信量4x

决策3: 是否使用共享专家 (Shared Experts)?

- 是 (DeepSeek): 共享专家处理通用知识, 路由专家处理特化知识
- 否 (Mixtral): 结构更简单, 但可能导致知识冗余

决策4: 负载均衡策略

- 辅助损失 (Auxiliary Loss): 经典方法, 但干扰主任务梯度
- Loss-Free (DeepSeek-V3): 偏置项调整, 无梯度干扰
- Expert Choice: 由专家选token, 天然均衡但无法预知路由

决策5: 并行策略 — EP度? 与TP/PP如何组合?

- EP (Expert Parallelism): 专家跨设备分布, All-to-All通信
- 混合: EP+TP (节点内) + PP (跨节点) -- 需匹配网络拓扑

决策6: 部署方式 — 全GPU vs 卸载 vs 量化?

- 全GPU: 最低延迟, 但需大量显存 (成本高)
- CPU/NVMe卸载: 显存需求低, 但延迟高, 需预测性预加载
- 量化 (INT4/INT8): 兼顾显存和延迟, 但需关注精度损失

> **关键要点:** MoE设计是一个多维trade-off空间: 专家数量、路由K值、负载均衡、并行策略、部署方式 -- 每个决策都相互关联, 需要整体优化。

6.3 MoE的核心挑战与机遇

背景: MoE正从"架构实验"转变为"生产必需": 2025年超过60%的开源模型采用MoE架构。这一转变带来了哪些挑战, 又蕴含哪些机遇?

核心挑战 (Challenges)

- 负载均衡仍然困难: 辅助损失干扰训练, Loss-Free方法尚未在所有场景验证
- 推理内存占用大: 全量参数需要加载, 即使大部分不使用
- 路由可解释性不足: 专家学到了什么? 路由决策能否解释?
- 训练不稳定: 路由崩塌 (Routing Collapse)、专家退化仍需处理
- All-to-All通信是分布式训练的瓶颈, 对网络带宽要求极高
- MoE模型的调试和诊断比Dense模型更复杂
- 量化MoE比量化Dense模型更困难 (专家间分布差异大)

重大机遇 (Opportunities)

- 硬件正在追赶: NVLink 900GB/s, CXL内存扩展正在解决通信瓶颈
- **2025年超过60%的开源模型发布采用MoE架构**
- **"架构实验" → "生产必需": 行业共识正在形成**
- Scaling Law明确支持: 同等计算预算下MoE优于Dense
- 多模态原生MoE (Gemini模式) 正在成为新趋势
- 动态MoE (自适应K, 动态专家数) 开辟新的设计空间
- MoE+RL推理 (DeepSeek-R1) 开辟高效推理新范式
- 开源生态繁荣: vLLM, SGLang, MegaBlocks等框架日趋成熟

关键要点: MoE已从实验性架构转变为生产必需品。挑战在于负载均衡和推理开销, 机遇在于硬件演进和60%+的行业采用率。

Based on 2025 open-source model release statistics and industry trends

思考题 (Discussion Questions)

以下问题涵盖MoE的算法、系统与部署三个维度，帮助深入理解核心trade-off:

Q1: 专家数量的Trade-off

- 为什么MoE模型中专家数量不是越多越好? 分析专家数量与模型性能、系统开销 (通信、内存、负载均衡难度) 之间的trade-off。

Q2: Token Choice vs Expert Choice的系统差异

- 比较Token Choice和Expert Choice两种路由方式在系统实现上的差异: 通信模式、负载均衡机制、容量缓冲 (Capacity Buffer) 的需求有何不同?

Q3: Loss-Free Balancing的优势

- DeepSeek-V3的Loss-Free Balancing为什么比传统Auxiliary Loss更好? 从梯度干扰 (Gradient Interference) 的角度分析其机制优势。

Q4: Mixtral 8x7B并行策略设计

- 在一个8-GPU集群上部署Mixtral 8x7B，设计并行策略 (EP/TP/DP的分配)，分析不同策略下的通信开销和显存占用。

Q5: All-to-All vs AllReduce

- MoE模型的Expert Parallelism中All-to-All通信与Dense模型的AllReduce有什么本质区别? 为什么MoE对网络带宽要求更高?