DeepSeek系列

DeepSeek-v1

论文：《DeepSeek LLM Scaling Open－Source Language Models with Longtermism》

链接：https://arxiv.org/pdf/2401.02954

DeepSeek-v1的架构参考LLaMA-2，主要区别是预训练数据规模更大，在层数上，7B版本采用30层网络，67B版本采用95层网络。FFN使用SwiGLU激活函数。

DeepSeek-v1的一个发现是数据和模型规模对模型的性能影响。

在微调过程中，发现了几个重要现象：

模型规模与微调策略的关系：

- 7B模型微调4个epoch后，而且在GSM8K和HumanEval数据集上可以继续提升性能
- 67B模型仅微调2个epoch就出现严重过拟合现象,且在GSM8K和HumanEval数据集上快速达到性能瓶颈

数学SFT数据的特殊性质：

- 随着数学SFT数据量增加,模型输出的重复率呈现上升趋势
- 这种现象主要源于数学SFT数据中存在相似的推理模式
- 较小规模的模型往往难以充分理解这些复杂的推理模式,因此容易产生重复性的响应

为进一步提升模型能力，他们采用DPO算法进行对齐。对齐设置：

- 学习率设置为5e-6
- 批处理大小为512
- 使用学习率预热和余弦学习率调度器
- 训练一个完整周期

得出结论，DPO能有效增强模型的开放式生成能力。

DeepSeek-V2

论文：《DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model》

链接：https://arxiv.org/pdf/2405.04434

DeepSeek-V2基于FFN MoE架构,使用MoE来代替原有的FFN，总参数量达236B，但每个token仅激活21B参数，支持128K的超长上下文。相比DeepSeek 67B，新版本在性能、效率和资源利用上都有显著提升：

- 训练成本降低42.5%
- KV缓存减少93.3%
- 最大吞吐量提升5.76倍

模型架构：

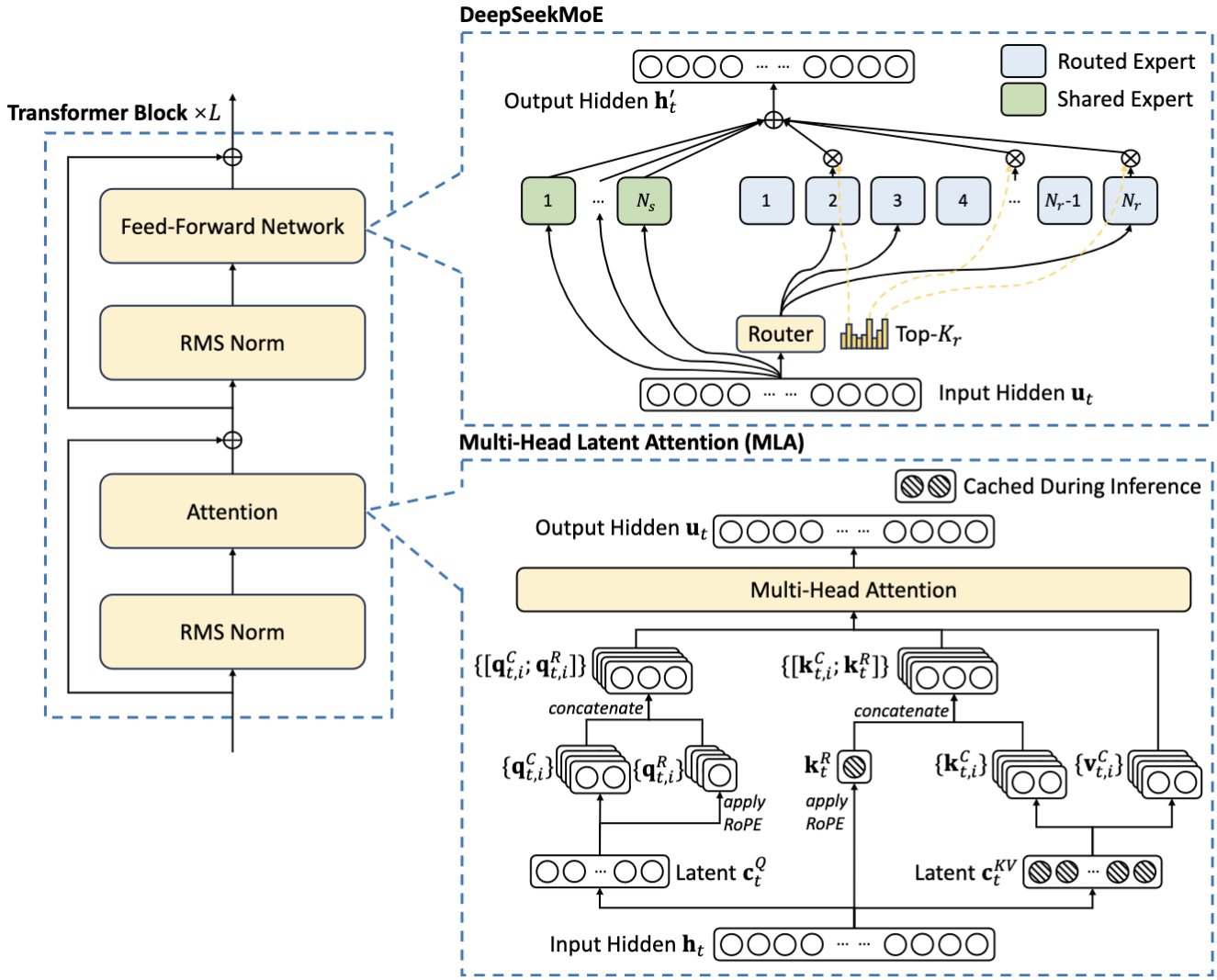

Figure 2 | Illustration of the architecture of DeepSeek-V2. MLA ensures efficient inference by significantly reducing the KV cache for generation, and DeepSeekMoE enables training strong models at an economical cost through the sparse architecture.

模型使用8.1T tokens进行预训练，并通过SFT和强化学习进一步优化。其最大的创新是提出了Multi-head Latent Attention(MLA)机制，通过将KV缓存压缩到隐向量空间来提升推理效率。

MLA机制详解

DeepSeek-V2用的MLA机制非常巧妙，在降低计算复杂度的同时，还能增加head的数量，从而提升模型的表达能力。

关于MLA的详细原理和代码实现，可以参考我之前的文章：

【LLM进阶系列】DeepSeek MLA 公式详细推导+代码实现

其他核心技术创新

1. 专家混合系统(DeepSeekMoE)

DeepSeek-V2采用了更细粒度的专家划分策略，在保持相同激活参数量的情况下，显著超越了传统MoE架构。其FFN层的计算过程如下：

- 专家分类：将专家分为"通才"(共享专家)和"专才"(路由专家)两类。
- 智能路由：每个专家都有一个特征向量$\mathbf{e}_i$，系统通过计算输入内容与特征的匹配度来选择最合适的专家。这就像根据任务特点为其匹配最适合的团队成员。

对于输入token $\mathbf{u}_t$：

$$
\begin{aligned}
\mathbf{h}_t' &= \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}\left(\mathbf{u}_t\right) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}\left(\mathbf{u}_t\right) \\
g_{i,t} &= \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}\left(\{s_{j,t} \mid 1 \le j \le N_r\}, K_r\right) \\ 0, & \text{otherwise} \end{cases} \\
s_{i,t} &= \mathrm{Softmax}_i\left(\mathbf{u}_t^{\top}\mathbf{e}_i\right)
\end{aligned}
\tag{1}
$$

其中：

- N_s, N_r：共享专家和路由专家的数量
- $\mathrm{FFN}_i^{(s)}$, $\mathrm{FFN}_i^{(r)}$：第i个共享/路由专家
- K_r：激活的路由专家数量
- $g_{i,t}$：专家门控值
- $\mathbf{e}_i$：路由专家中心向量
- $s_{i,t}$：token-expert亲和度

2. 三层均衡机制(Auxiliary Loss for Load Balance)

负载不平衡会带来两个主要问题:一是容易导致路由坍缩,使部分专家无法得到充分训练;二是在专家并行时会降低计算效率。为了解决这些问题,DeepSeek-V2设计了三种辅助损失函数,分别用于控制:

1. 专家级别负载均衡 $\mathcal{L}_{\mathrm{ExpBal}}$ - 确保专家工作量均衡
2. 设备级别负载均衡 $\mathcal{L}_{\mathrm{DevBal}}$ - 保证设备负载均衡
3. 通信均衡 $\mathcal{L}_{\mathrm{CommBal}}$ - 优化设备间通信

1. 专家级均衡损失

这个损失函数主要用来避免路由坍缩问题,公式如下:

$$
\begin{aligned}
\mathcal{L}_{\mathrm{ExpBal}} &= \alpha_1 \sum_{i=1}^{N_r} f_i P_i \\
f_i &= \frac{N_r}{K_r T} \sum_{t=1}^{T} 1(\text{ Token t selects Expert i}) \\
P_i &= \frac{1}{T} \sum_{t=1}^{T} s_{i,t}
\end{aligned}
\tag{2}
$$

其中:

- α_1 是控制这个损失权重的超参数
- f_i 表示第i个专家实际被选中的频率
- P_i 表示第i个专家被选中的预期概率
- $1(\cdot)$ 是指示函数,当条件成立时为1,否则为0
- K_r 是每个token可以选择的专家数量
- T 是序列中token的总数

大概解释一下计算过程:

1. f_i 是计算每个专家实际被选中的频率:

- $\frac{N_r}{K_r T}$ 是一个归一化因子,其中:
  - N_r 是路由专家总数
  - K_r 是每个token可以选择的专家数量
  - T 是序列长度
- $\sum_{t=1}^{T} 1(\text{Token t selects Expert i})$ 统计专家i被选中的次数

2. P_i 计算每个专家被选中的预期概率:

- $\frac{1}{T} \sum_{t=1}^{T} s_{i,t}$ 对所有token的专家选择概率求平均
- $s_{i,t}$ 是token t选择专家i的概率分数

3. 最终的损失 $\mathcal{L}_{\text{ExpBal}}$ 通过:

- 对所有专家的实际频率和预期概率的乘积求和
- 乘以权重系数 α_1 控制损失强度

这个损失函数的目的是:

- 鼓励专家被均匀选择,避免某些专家被过度使用或闲置
- 通过实际频率和预期概率的对比来调节路由行为
- 帮助模型学习更合理的专家分配策略

2. 设备级均衡损失

为了确保计算负载在不同设备间均衡分布,模型将专家分成 D 组 $\{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_D\}$,每组部署在一个设备上。设备级均衡损失计算如下:

$$
\begin{aligned}
\mathcal{L}_{\text{DevBal}} &= \alpha_2 \sum_{i=1}^{D} f_i' P_i' \\
f_i' &= \frac{1}{\mathcal{E}_i} \sum_{j \in \mathcal{E}_i} f_j \\
P_i' &= \sum_{j \in \mathcal{E}_i} P_j
\end{aligned}
\tag{3}
$$

让我们详细解释这个公式:

1. 首先,模型将所有专家分成D个组,每组专家 $\mathcal{E}_i$ 部署在一个设备上

2. 对于每个设备i:
    - f_i' 计算该设备上所有专家的平均实际使用频率:
        - $\sum_{j \in \mathcal{E}_i} f_j$ 对设备i上所有专家的实际频率求和
        - 除以 $\mathcal{E}_i$ (该设备上专家的数量)得到平均值
    - P_i' 计算该设备上所有专家的预期使用概率总和:
        - $\sum_{j \in \mathcal{E}_i} P_j$ 将设备i上所有专家的预期概率相加
3. 最终的设备级均衡损失通过:
    - 对所有D个设备的 $f_i' P_i'$ 求和
    - 乘以权重系数 α_2 来控制这个损失的强度

这个损失函数的目标是使每个设备上的专家组获得均衡的工作量,避免某些设备过载而其他设备闲置的情况。

3. 通信均衡损失
通信均衡损失的目的是避免某些设备接收过多的tokens导致通信瓶颈。让我们详细解释这个公式:

$$
\begin{aligned}
\mathcal{L}_{\text{CommBal}} &= \alpha_3 \sum_{t=1}^{D} f_i'' P_i'' \\
f_i'' &= \frac{D}{MT} \sum_{t=1}^{T} 1(\text{ Token } t \text{ is sent to Device i}) \\
P_i'' &= \sum_{j \in \mathcal{E}_i} P_j
\end{aligned}
\tag{4}
$$

公式各部分的含义如下:

1. f_i'' 表示设备i接收tokens的实际频率:
    - $\sum_{t=1}^{T} 1(\text{ Token } t \text{ is sent to Device i})$ 统计发送到设备i的token总数
    - 乘以 $\frac{D}{MT}$ 进行归一化,其中D是设备数,M是每个设备可发送的token上限,T是序列长度
2. P_i'' 表示设备i的预期接收概率:
    - $\sum_{j \in \mathcal{E}_i} P_j$ 将设备i上所有专家的预期使用概率相加
3. 最终的通信均衡损失:
    - 对所有D个设备的 $f_i'' P_i''$ 求和
    - 乘以权重系数 α_3 控制损失强度

这个损失函数通过以下方式优化通信:

- 限制每个设备最多发送MT个hidden states
- 鼓励每个设备接收大约MT个hidden states
- 实现设备间通信负载的均衡分布

3. 智能资源管理

DeepSeek-V2引入了灵活的资源管理策略：

- 设备受限路由：限制每个任务最多使用3个设备，在保持性能的同时降低通信成本
- Token丢弃策略：根据计算预算动态调整处理的token数量，类似于项目中的任务优先级管理
- 保留10%的训练序列不进行丢弃，确保模型行为的一致性

4. 模型优化与扩展

- 采用YaRN技术将上下文长度扩展到128K，显著提升了模型处理长文本的能力
- 通过SFT和GRPO进行模型对齐，使模型输出更符合人类期望

这些创新不仅提升了模型性能，还大幅降低了训练和推理成本，使DeepSeek-V2在实际应用中更具优势。

DeepSeek-V3

报告：《DeepSeek-V3 Technical Report》

链接：https://arxiv.org/pdf/2412.19437

DeepSeek-V3的MLA和FFN MoE架构与DeepSeek-V2相同，但是对FFN MoE的平衡策略进行了更新。

FFN MoE架构

无辅助损失的负载平衡

DeepSeek-V3将专家划分为更细粒度,在相同激活专家参数的情况下,DeepSeekMoE性能显著超越传统MoE架构。

对于输入token $\mathbf{u}_t$,FFN的输出计算如下:

$$
\begin{aligned}
\mathbf{h}_t' &= \mathbf{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}\left(\mathbf{u}_t\right) + \sum_{i=1}^{N_r} g_{i,t} \mathrm{FFN}_i^{(r)}\left(\mathbf{u}_t\right), \\
g_{i,t} &= \frac{g_{i,t}'}{\sum_{j=1}^{N_r} g_{j,t}'}, \\
g_{i,t}' &= \begin{cases} s_{i,t}, & s_{i,t} \in \operatorname{Topk}\left(\left\{s_{j,t} \mid 1 \leqslant j \leqslant N_r\right\}, K_r\right), \\ 0, & \text{otherwise}, \end{cases} \\
s_{i,t} &= \operatorname{Sigmoid}\left(\mathbf{u}_t^T \mathbf{e}_i\right),
\end{aligned}
\tag{5}
$$

与V2相比,主要区别在于:

1. 门控值计算方式改变,用Sigmoid替代了Softmax
2. 增加了归一化步骤,确保门控值之和为1
3. 引入了共享专家和路由专家的概念

其中:

- N_s, N_r：共享专家和路由专家的数量
- $\mathrm{FFN}_i^{(s)}$, $\mathrm{FFN}_i^{(r)}$：第i个共享/路由专家

- K_r：激活的路由专家数量
- $g_{i,t}$：归一化后的专家门控值
- $\mathbf{e}_i$：路由专家中心向量
- $s_{i,t}$：token-expert亲和度

为避免V2中辅助损失带来的梯度干扰问题,V3采用无辅助损失的负载平衡策略:

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \operatorname{Topk}\left(\{s_{j,t} + b_j \mid 1 \leqslant j \leqslant N_r\}, K_r\right) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

通过动态调整偏置项 b_i 来实现负载平衡,避免了显式的辅助损失。

这里的偏置项 b_i 是一个动态调整的参数,用于平衡专家的负载。当某个专家被过度使用的时候，对应的b_i会被减小，降低该专家被选中的概率。某个专家使用不足时:，其对应的 b_i 会增。提高该专家被选中的概率。

互补序列级辅助损失

为防止单个序列内的某个专家重要度过高，极端不平衡，V3引入了序列级辅助损失:

$$\begin{aligned} \mathcal{L}_{\text{Bal}} &= \alpha \sum_{i=1}^{N_r} f_i P_i \\ f_i &= \frac{N_r}{K_r T} \sum_{t=1}^{T} 1\left(s_{i,t} \in \operatorname{Topk}\left(\{s_{j,t} \mid 1 \leqslant j \leqslant N_r\}, K_r\right)\right) \\ s'_{i,t} &= \frac{s_{i,t}}{\sum_{j=1}^{N_r} s_{j,t}} \\ P_i &= \frac{1}{T} \sum_{t=1}^{T} s'_{i,t} \end{aligned} \tag{7}$$

其中:

- f_i 表示专家i的实际使用频率
- P_i 表示专家i的预期使用概率
- α 为权重系数
- T 为序列长度

可以这么理解，序列中每个token都会独立选择专家来处理。

举个例子:
假设有一个序列"我喜欢机器学习":

- "我"这个token可能选择专家1,3
- "喜欢"这个token可能选择专家2,4
- "机器"这个token可能选择专家1,5
- "学习"这个token可能选择专家3,4

但我们希望在整个序列维度上，专家的使用要相对均衡。

关于序列级辅助损失公式:

公式中的P_i反映了专家i在整个序列中的"理论重要性"，f_i反映了专家i实际被选中的频率。

当两者不一致时，说明模型可能过度依赖某些专家，需要通过最小化损失来调整。

这种设计有效地避免了专家使用过于集中的问题，因为：

- 如果所有token都选择相同的专家，那么f_i和P_i的分布会很接近，损失会变大
- 如果专家选择更加分散，那么f_i和P_i的分布差异会增大，损失会变小

与V2的通信均衡损失相比：

1. 关注点从设备间通信转向序列内专家使用的均衡
2. 计算粒度从设备级别细化到专家级别
3. 损失函数形式更简单，不需要考虑设备约束

其他优化策略

1. 节点限制路由:继续使用V2的策略,限制每个任务最多使用3个设备
2. Token丢弃:放弃V2的动态Token丢弃策略,简化训练过程

多Token预测(MTP)

MTP架构

MTP的目标是一次预测多个token,类似投机采样,可以显著减少解码时的访存压力。在准确率超过85%时,可实现1.8倍的解码速度提升。

MTP使用K个顺序模块预测K个额外token,每个模块包含:

- 共享的特征层 $\mathrm{Emb}(\cdot)$
- 共享的输出头 $\mathrm{OutHead}(\cdot)$
- Transformer块 $TRM_i(\cdot)$
- 投影矩阵 W_i

对于第i个token的预测,计算过程为:

$$\mathbf{h}_i'^k = M_k \left[\mathrm{RMSNorm}\left(\mathbf{h}_i^{k-1}\right); \mathrm{RMSNorm}\left(\mathrm{Emb}\left(t_{i+k}\right)\right)\right] \tag{8}$$

$$\mathbf{h}_{1:T-k}^k = \mathrm{TRM}_k\left(\mathbf{h}_{1:T-k}'^k\right) \tag{9}$$

$$P_{i+k+1}^k = \mathrm{OutHead}\left(\mathbf{h}_i^k\right) \tag{10}$$

MTP训练目标

MTP的训练使用交叉熵损失:

$$\mathcal{L}_{\mathrm{MTP}}^k = \mathrm{CrossEntropy}\left(P_{2+k:T+1}^k, t_{2+k:T+1}\right) = -\frac{1}{T}\sum_{i=2+k}^{T+1} \log P_i^k\left[t_i\right] \tag{11}$$

最终的MTP损失为所有深度的平均:

$$\mathcal{L}_{\text{MTP}} = \frac{\lambda}{D} \sum_{k=1}^{D} \mathcal{L}_{\text{MTP}}^{k} \tag{12}$$

推理部署

MTP模块在推理时具有灵活性:

1. 可以直接丢弃,不影响主模型功能
2. 可用于推测性解码,提升生成速度
3. 主模型可独立正常工作