

1. PPO

视频详解：【中文语音】Umar Jamil 讲解RLHF和PPO https://www.bilibili.com/video/BV1QCq5Y8Eto?vd_source=9d018d6ed6982648faec801ea28f74d9

我们可以暂时地将强化学习RL的过程简单地认为是“采样+求梯度”的过程。对策略的采样+对采样到策略的期望求梯度（求导）。

对于采样，为什么LLM微调可以使用的强化学习策略更新方法？因为 model 基于当前句子生成下一个 token，再基于新的句子生成下下步的 token；等同于策略 π 基于当前状态 state 生成下一个动作 action，再基于新的状态 state 生成下下个动作 action。

而最原始的RL策略更新公式是对策略获取奖励的期望求梯度，也就是采样所有的可能的路径。在LLM的微调中不可能采样所有可能的结果（LLM如此巨大的词表），因此一般是使用不同的温度系数/top k等采样几组有限的结果，然后沿着梯度方向小调整。

具体而言，准备一个问题库和答案库，使用LLM为每个问题生成一系列的答案，这就构成了一系列的生成路径，每个路径都有对数概率（和强化学习策略一样，因为LLM会为每个可能的下一个词计算概率），我们就可以计算这些状态对数概率的梯度。

一个不同的地方是，正常LLM会对每个可能的预测输出应用softmax，但由于RL公式要求的是对数概率，所以需要在这里应用对数形式的softmax。

我们并不需要这个句子中这个位置所有可能的候选token的对数概率，只需要这个位置实际选择的token的对数概率。这样就能得到这条轨迹中每个状态动作各个位置的对数概率。

有了这些信息之后让pytorch自动计算梯度，然后用奖励模型给出的奖励值乘以每个梯度：

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau),$$

如何计算奖励值？假设奖励模型是一个transformer，顶层只有一个输出特征的线性层。奖励模型输出的对应位置的奖励，分别对应轨迹中的每个动作。我们就可以把这些奖励值加起来，得到整个过程的总奖励 $R(T)$ 。

而这种办法是采样有限的轨迹的奖励均值来近似所有可能轨迹的期望。当然我们无法轻易增加采样数量，LLM运行成本很高，我们就需要用其他方法来让我们的梯度方向接近真实的梯度方向。所以我们要做的就是降低估计器的方差。

第一种方法是：当前我们策略状态行为所产生的奖励，不会影响到过去策略状态行为所产生的奖励（不会改变我们已获得的奖励）。因此当我们计算T时刻的奖励时，我们不会考虑该动作发生之前的所有奖励。我们不再从最初时刻开始计算整个轨迹的奖励，我们可以从执行动作的那一刻开始计算奖励。这个术语被称为“未来累积奖励”——从当前状态开始能获得的总奖励。

为什么要这样计算剩余轨迹的奖励？因为当前表达式其实是对真实期望值的一种近似，这种情况下项越少越好，如果我们不考虑这些因素，就能避免在近似计算中引入额外干扰。因此简化表达式对方差更有利。

第二种方法是：引入baseline，可以引入一个强化学习研究证明。引入常数可以降低方差，当然这个值也不一定是常数，他也可以是与状态相关的变量，也可以是一个状态函数。

$$\nabla_{\theta}(J(\theta)) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) \left(\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'}) - b \right)$$

Baseline
Can also be dependent upon the state

我们要计算的是轨迹的累积奖励，所以我们将每个对数概率 $\log \pi_{\theta}(a_{i,t} | s_{i,t})$ 乘以一个表示未来奖励的系数，即从当前状态行为到轨迹终点的累积奖励。再减去一个基线值（可以是一个与状态相关的函数），就是价值函数。

价值函数告诉我们在某种策略下特定状态的价值。它表示从某个起始状态出发能获得的预期回报，并在之后的整个过程中按照既定策略行动。

例如，在强化学习走格子任务中，奖励目标附近的位置价值较高，因为很有可能模型在这里下一步直接走到奖励目标上。在惩罚目标附近的位置价值很可能小于远离惩罚目标的格子，因为两个位置中，距离惩罚目标最近的格子更有可能在接下来的行动中受到惩罚。

这就是价值函数的含义。价值函数告诉我们从某个状态开始，按照策略行动，能获得多少预期回报。

那么如何估算这个价值函数？

和奖励模型类似，我们可以创建一个神经网络，在其顶部添加一个线性层来估算价值函数。实际过程中我们会使用相同的语言模型，在我们要优化的模型顶部再加一个线性层（除了用于生成词汇的那一层外，我们还添加另一层来估算价值函数），这样Transformer层参数可共用。

这样我们可以估算每个状态（LLM预测句子中每个位置）的价值，甚至是整个序列的总体价值。

回顾一下，我们将整个轨迹的奖励转换为未来累积奖励（不是从时刻0开始，而是从我们正在考虑的动作状态开始），我们还发现可以引入一个与状态相关的基准线，这样做不会影响近似结果，因此这个方法仍然保持无偏性。就是说它平均而言会收敛于真实梯度，但波动会更小。

在强化学习中，这个表示未来累积奖励的函数：

$$i,t | s_{i,t}) \left(\underbrace{\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'})}_{\text{Should remind you of the } \mathbf{Q} \text{ function.}} \right)$$

$$Q^{\pi}(s, a) = E_{s'}[r(s, a) + \gamma E_{a'}[Q^{\pi}(s', a')]]$$

也被称为Q函数，Q函数告诉我们从某个状态出发，采取特殊行动后预期能获得多少奖励。可以把表达式简化成状态的Q函数以及t时刻的动作减去t时刻的值，两者的差就是所谓的优势函数（advantage function）：

$$s_{i,t}) \left(\underbrace{Q^{\pi}(s, a) - V^{\pi}(s)}_{\text{Known as the } \mathbf{advantage} \text{ function}} \right)$$

Q函数表示从状态t开始采取行动a，之后按策略行动能获得的预期回报。而价值函数则表示从状态s出发（按照策略行动）能获得预期回报是多少。两者之差，优势函数告诉我们某个特定动作的优势程度，与在状态s中可选的一般动作相比。

$$\begin{aligned} \nabla_{\theta}(J(\theta)) &\approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) \underbrace{(Q^{\pi}(s_t, a_t) - V^{\pi}(s_t))}_{\text{advantage}} \\ \nabla_{\theta}(J(\theta)) &\approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) \underbrace{A^{\pi}(s_t, a_t)}_{\text{advantage}} \end{aligned}$$

优势函数反映了某个动作相对于平均水平的优势程度，即考虑该动作相对其他可选动作的优势。如果我们解读这个表达式，它实际上是在指导模型对于特定状态下每个动作的对数概率，我们需要将其乘以相应的优势值，这个梯度会指明一个方向，我们再使用梯度上升法来优化参数。

简而言之我们是在强制策略提高某些动作的概率，也就是增加高优势动作的对数概率。也就是说他们能带来高于平均水平的回报，同时降低每种状态下那些表现不佳行为的概率。

假设我们只计算第一次近似值，因为我们用价值函数来估算大部分轨迹，这会导致结果有较大偏差。因为我们估算出的值会不太准确，我们用价值函数估算大部分轨迹而价值函数本身也是一个近似值。

或者我们可以引入来自实际轨迹的更多奖励来改进这个近似，然后只用价值函数对轨迹的一小部分做近似。或者用获得的奖励来近似整个轨迹，完全不使用实际价值头来近似。

但使用更多项会导致更高的方差，使用更少项用价值函数直接估计会导致更多的偏差。我们可以采用一种称为广义优势估计的方法，这种方法本质上是计算所有这些项的加权和。

we can estimate the advantage term in many ways, as follows:

$$\begin{aligned} \hat{A}^\pi(s_t, a_t) &= [r(s_t, a_t) + \gamma V^\pi(s_{t+1})] - V^\pi(s_t) \\ \hat{A}^\pi(s_t, a_t) &= [r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 V^\pi(s_{t+2})] - V^\pi(s_t) \\ \hat{A}^\pi(s_t, a_t) &= [r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \gamma^3 V^\pi(s_{t+3})] - V^\pi(s_t) \end{aligned}$$

If we stop too early, we will get a very **high bias** (because we are approximating the value function, and only using one "real" reward from our trajectory).
If we stop after many terms, we will get **high variance**. In order to solve this **bias-variance problem**, we can take the weighted sum of the terms to obtain the **Generalized Advantage Estimation**:

$$\delta_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

$$\hat{A}_t = \delta_t + \gamma \lambda \hat{A}_{t+1}$$

This is a **recursive formula** that for the last term equals the first expansion
For the second-last term equals to the second expansion (weighted by λ)
Etc.

梯度表达式中，每个对数概率都乘以了优势函数：

$$\nabla_\theta (J(\theta)) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) A^\pi(s, a)$$

现在的问题是，我们在采样决策路径时，我们需要从LLM中在每次梯度上升时进行采样。

在这个优化过程中的每一步骤，需要计算大量的路径，需要计算大量优势函数，需要计算大量的奖励值，还要计算大量的对数概率。这个过程会变的非常低效，优化中每前进一步都需要大量的重新计算。

然而我们还记得，整个表达式实际上是对期望的近似，那么我们就可以引入重要性采样：

$$\begin{aligned} \underline{E_{x \sim p(x)}[f(x)]} &= \int p(x) f(x) dx \\ &= \int \frac{q(x)}{q(x)} p(x) f(x) dx \\ &= \int q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= E_{x \sim q(x)} \left[\frac{p(x)}{q(x)} f(x) \right] \end{aligned}$$

我们可以对梯度表达式应用相同的方法：

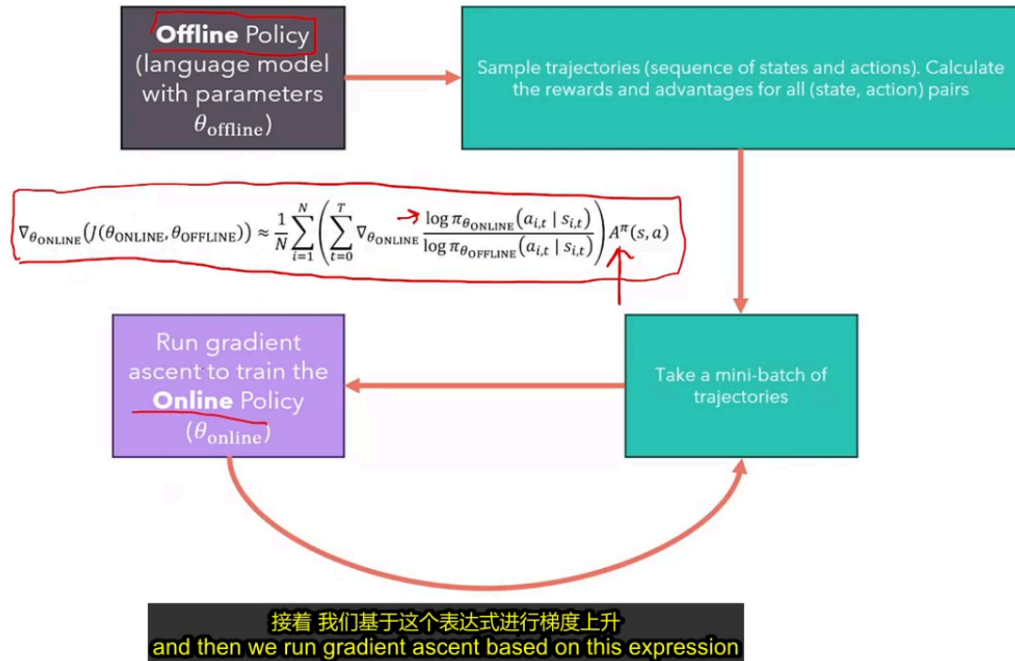
$$\nabla_{\theta}(J(\theta)) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \right) A^{\pi}(s, a) \longrightarrow \nabla_{\theta_{\text{ONLINE}}}(J(\theta_{\text{ONLINE}}, \theta_{\text{OFFLINE}})) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^T \nabla_{\theta_{\text{ONLINE}}} \frac{\log \pi_{\theta_{\text{ONLINE}}}(a_{i,t} | s_{i,t})}{\log \pi_{\theta_{\text{OFFLINE}}}(a_{i,t} | s_{i,t})} \right) A^{\pi}(s, a)$$

Trajectories sampled from π_{θ} Trajectories sampled from $\pi_{\theta_{\text{OFFLINE}}}$

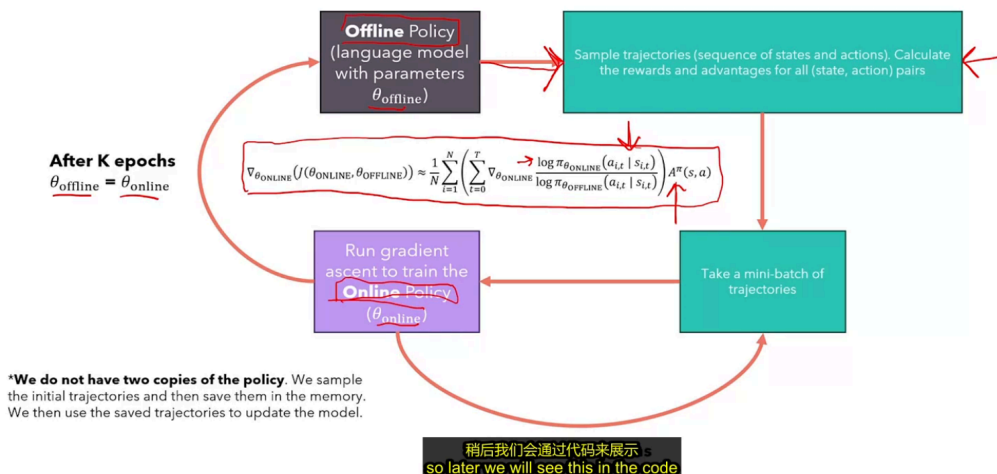
我们在这里从某个策略进行采样，但我们可以用重要性采样来优化它，从另一个策略采样（假设它是一个不同的神经网络）。采样过程就是根据给定问题生成相应文本，实际上就是从神经网络中进行采样。

对每一个项也就是每一个优势值，我们不只是将其乘以我们正在优化的网络给出的概率，还要将其除以 $Q(x)$ ， $Q(x)$ 是我们采样分布的对数概率，我们把这个用于采样的分布，称为离线策略 π 。

而我们要优化的是在线策略分布 π 。通过重要性采样，我们可以利用一个网络的采样来计算另一个网络的期望值，同时优化另一个网络。这被称为离线学习。



Off-Policy learning



无需每次都从正在优化的策略中采样，我们只需从待优化的网络中采样一次，然后将轨迹保存在内存中，对网络进行多步优化，优化几步后我们就可以采样新的轨迹了

PPO loss:

The PPO loss

$$L_{\text{POLICY}} = \min\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t, \text{clip}\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon\right) \hat{A}_t\right)$$

$$L_{\text{VF}} = \frac{1}{2} \left\| V_{\theta(s)} - \left(\sum_{t=0}^T \gamma^t r_t \mid s_0 = s \right) \right\|_2^2$$

$$L_{\text{ENTROPY}} = - \sum_x p(x) \log p(x)$$

$$L_{\text{PPO}} = L_{\text{POLICY}} + c_1 L_{\text{VF}} + c_2 L_{\text{ENTROPY}}$$

PPO损失计算的是 $\min((\text{在线策略/离线策略}) * \text{优势函数}, \text{裁剪函数})$ 。裁剪函数将表达式（在线策略/离线策略）限制在1加减epsilon中。

我们可以看到，裁剪函数限制的是在线策略和离线策略采样的对数概率的比值。

如果在优化过程中，某个动作的对数概率 远高于 我们采样时的概率。这就意味着我们在增加未来选择该动作的概率，我们不希望这种增幅过大，因此我们要将其裁剪到最大值。

另一方面，如果我们想要降低某个动作的概率，相比之前的水平，我们不希望它减少过多。

这意味着在优化过程中，我们在调整行为概率，也就是在特定提示下选择特定词元的可能性，我们在不断地调整这些概率，但又不希望变化太大，想要循序渐进。

为什么？因为如果调整太大，模型可能就无法充分探索其他可能性了。这可能导致模型过分关注某一行为，要么总是避免，要么频繁使用。我们不希望这个变化太大，这就是我们使用两者最小值 $\min(\dots)$ 的原因，我们要尽可能做出最保守的更新。

当然，我们需要训练来使得神经网络能够逼近价值函数，因此我们在PPO损失函数中引入了另一个项 L_{VF}

这是用来训练价值函数估计器的，简单来说它就是指，根据特定状态模型会输出一个估计值。然后我们把它和该状态的实际价值作比较。这个实际价值是根据我们采样的行动序列得出的。

我们有多个行动序列，每个序列有多个状态-动作对组成，每对都有相应的奖励。因此我们可以计算出该状态的实际价值。

$$L_{\text{PPO}} = L_{\text{POLICY}} + c_1 L_{\text{VF}} + c_2 L_{\text{ENTROPY}}$$

PPO损失由策略优化项 + 评估价值函数准确性的损失函数 + 熵损失用于增加模型的不确定性。

假设模型缺少最后一项熵损失，它可能会单纯地优化行为，以特定方式来选择行为。例如更频繁地选择那些带来显著优势的行为，而那些低于平均优势的行为，会被较少的选择。这样依赖模型在选择词语时就会变得非常固化。模型会一直选择那些带来好处的词语，完全不采用那些效果不佳的词语。

既然我们需要最大化PPO，那么我们就需要最大化policy目标函数，同时最小化 L_{VF} 损失函数，最后还要最大化熵值以增加更多不确定性。

然而，我们直接使用PPO损失函数，模型可能会学到一些特定的词或短语。或者是一些总能获得高分的固定用语，模型可能会不断选择这些用语来获得高分，但这对人来说可能毫无意义。

比如我们会要求模型回答的有礼貌，这时我们问模型上海在哪里，他可能一直回答说谢谢，那么奖励模型很可能会给这种回答一个高分，因为这很有礼貌，但这对人类来说毫无意义。因此我们希望模型能生成有意义的回答，这些回答应该与训练数据相似。

所以我们要限制模型不能只追求高分，但同时还要生成与原本相似的答案。即只看未训练或未对齐的模型，因此我们复制了一份模型，用于优化并冻结其权重。

这就是所谓的冻结模型，我们为其每个步骤生成奖励，即轨迹中的每一步，但会根据每步对数概率的变化程度进行惩罚，所以对于每个隐藏状态，我们可以用之前提到的线性层来生成单输出特征的奖励。

于此同时，对于每个隐藏状态，我们还会计算对数概率（用另一个线性层生成logits）。对冻结模型也进行相同的操作。最后调整奖励值。在这个特定时刻的奖励，我们定义它等于初始时刻的奖励，减去冻结模型与当前模型对数概率之间的KL散度。

具体来说，我们比较冻结模型的对数概率与 我们正在优化的模型策略的对数概率，我们要对模型生成答案进行约束，避免它与冻结模型相差太远。

因此我们的目标是最大化奖励，但同时我们也要防止模型钻空子，不能仅仅因为生成任何输出就得到奖励。我们希望模型能真正获得有意义的奖励。即与未优化时的输出相似的高质量答案。

2. 从 PPO 到 GRPO

论文原文: [DeepSeekMath: 在开放语言模型中突破数学推理的极限](#)

论文精读: https://www.bilibili.com/video/BV1qUFMeGE2q/?share_source=copy_web

2.1 GRPO的提出

Proximal Policy Optimization (PPO) 被广泛用于LLM的强化学习微调 (RLHF) 阶段。在RLHF中，PPO通过最大化以下代理目标函数来优化LLM：

$$\mathcal{J}_{PPO}(\theta) = \mathbb{E}_{q \sim P(Q), o \sim \pi_{\theta_{old}}(O|q)} \left[\frac{1}{|o|} \sum_{t=1}^{|o|} \min \left(\frac{\pi_{\theta}(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})} A_t, \text{clip} \left(\frac{\pi_{\theta}(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right]$$

π_{θ} ：当前策略模型 (Policy Model) 。

$\pi_{\theta_{old}}$ ：旧的策略模型（上一次迭代的模型），用于计算重要性采样比率。

q ：从问题数据集 $P(Q)$ 中采样的问题 (prompt) 。

o ：在旧策略 $\pi_{\theta_{old}}$ 下生成的输出。

ϵ ：PPO 引入的裁剪超参数 (clipping-related hyperparameter)，用于稳定训练。

A_t^{**} ：优势函数 (Advantage Function)，用于衡量当前策略比基准策略更优的程度，通过广义优势估计 (GAE, Schulman et al., 2015) 计算：

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} (r_{t'} - V_{\psi}(s_{t'}))$$

因此，在 PPO 中，价值函数需要与策略模型一起训练。此外，为了防止策略过度优化奖励模型，标准方法是在每个 token 的奖励中加入一个来自参考模型 (Reference Model) 的 KL 惩罚：

$$r_t = r_{\phi}(q, o_{\leq t}) - \beta \log \frac{\pi_{\theta}(o_t|q, o_{<t})}{\pi_{\text{ref}}(o_t|q, o_{<t})}$$

r_{ϕ} ：奖励模型 (Reward Model)，用于计算模型生成结果的奖励。

π_{ref} : **参考模型 (Reference Model)** , 通常是 **初始 SFT (Supervised Fine-Tuning) 模型**, 用于作为对比基准。

π_{θ} : **当前策略模型 (Policy Model)** , 用于生成 token 。 β : KL 惩罚的系数, 控制 KL 正则项的影响程度。

β : KL 惩罚的系数, 控制 KL 正则项的影响程度。

然而:

1.PPO中采用的 Value Model (价值函数) 通常是另一个与策略模型大小相当的模型, 会带来大量内存和计算负担。

2.在RL训练期间, 价值函数的主要作用是计算 Advantage (优势函数) 时减少梯度估计的方差, 提高训练稳定性。

$$A(s, a) = Q(s, a) - V(s)$$

其中 $Q(s, a)$ 表示采取动作 a 之后的预期总回报 (通常来自蒙特卡洛采样) ;

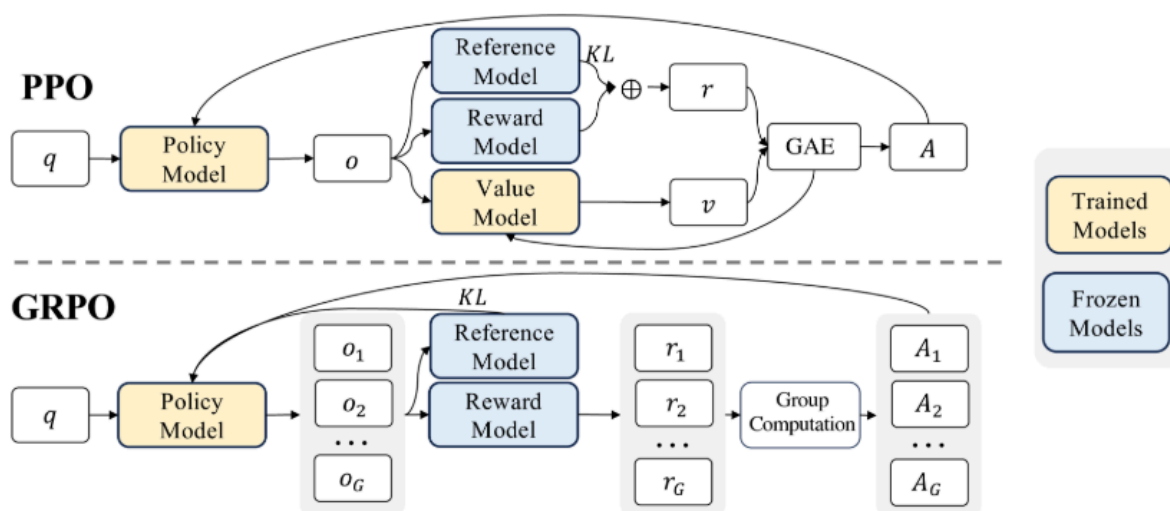
$V(s)$ 是价值函数, 表示该状态的预期回报, 作为基准 (baseline) 。

3.由于LLM训练中奖励函数的特殊性, 奖励通常只作用于最后一个 token, 而 PPO 需要对整个 token 序列估计价值函数, 这会导致价值函数训练变得困难, 因为前面 token 的奖励信息是间接的。

在 LLM 的强化学习 (RLHF, Reinforcement Learning from Human Feedback) 中, 奖励通常只针对最终生成的整个序列, 或者仅针对最后一个 token。

但是价值函数的目标是预测每个 token 的回报, 也就是说:

- 在 RL 典型任务 (如游戏) 中, 每个状态 s_{ss} 都有一个合理的 $V(s)$ 。
- 但在 LLM 中, 如果只有最终 token 才有奖励, 那么训练价值函数 $V(s)$ 变得困难, 因为前面的 token 并没有明确的奖励信号。



图注: PPO 和我们的 GRPO 示范。GRPO 放弃了价值模型, 而是根据组分数估计基线, 从而显著减少了训练资源

因此我们提出了组相对策略优化 Group Relative Policy Optimization (GRPO) , 它消除了像 PPO 中那样需要额外的价值函数近似, 而是使用多个采样输出的平均奖励。更具体地说, 对于每个问题 q 中, GRPO对一组输出从旧策略 π_{θ} 进行采样 $\{o_1, o_2, \dots, o_G\}$, 然后通过最大化以下目标来优化策略模型:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E} [q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[\frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left(\frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{KL}[\pi_{\theta} || \pi_{\text{ref}}] \right\}$$

其中：

- π_{θ} 和 $\pi_{\theta_{old}}$ 分别是当前和旧的策略模型；
- q 是输入问题，从问题分布 $P(Q)$ 采样；
- o_i 是采样得到的输出，每组有 G 个输出；
- $\hat{A}_{i,t}$ 是 GRPO 计算的优势估计（基于组内相对奖励）；
- ϵ 是 PPO 的裁剪超参数，控制策略更新幅度；
- β 是 KL 散度的正则化系数；
- $\mathbb{D}_{KL}[\pi_{\theta} || \pi_{\text{ref}}]$ 是当前策略 π_{θ} 和参考策略 π_{ref} 之间的 KL 散度。

Algorithm 1 Iterative Group Relative Policy Optimization

Input initial policy model $\pi_{\theta_{\text{init}}}$; reward models r_{φ} ; task prompts \mathcal{D} ; hyperparameters ϵ, β, μ

- 1: policy model $\pi_{\theta} \leftarrow \pi_{\theta_{\text{init}}}$
- 2: **for** iteration = 1, ..., I **do**
- 3: reference model $\pi_{\text{ref}} \leftarrow \pi_{\theta}$
- 4: **for** step = 1, ..., M **do**
- 5: Sample a batch \mathcal{D}_b from \mathcal{D}
- 6: Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_{\theta}$
- 7: Sample G outputs $\{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(\cdot | q)$ for each question $q \in \mathcal{D}_b$
- 8: Compute rewards $\{r_i\}_{i=1}^G$ for each sampled output o_i by running r_{φ}
- 9: Compute $\hat{A}_{i,t}$ for the t -th token of o_i through group relative advantage estimation.
- 10: **for** GRPO iteration = 1, ..., μ **do**
- 11: Update the policy model π_{θ} by maximizing the GRPO objective (Equation 21)
- 12: Update r_{φ} through continuous training using a replay mechanism.

Output π_{θ}

与标准 PPO 不同，GRPO 计算优势估计时，仅基于 **同一问题下不同输出的相对奖励**，更符合奖励模型的 **比较性质**，因为奖励模型通常是基于不同答案的相对评分进行训练的。此外，GRPO **直接在损失函数中加入 KL 散度项**，而不是像 PPO 那样在奖励函数中加 KL 惩罚，避免影响优势估计的计算。

我们使用以下无偏估计器来估计 KL 散度：

$$\mathbb{D}_{KL}[\pi_{\theta} || \pi_{\text{ref}}] = \frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta}(o_{i,t}|q, o_{i,<t})} - \log \frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta}(o_{i,t}|q, o_{i,<t})} - 1$$

该估计方式保证了 KL 散度始终为正值；通过无偏估计，可以更稳定地约束策略更新，防止过度优化。

GRPO 相较于 PPO 的核心区别

优势估计方式不同：

- PPO 直接使用 单个样本的奖励 计算优势估计；
- GRPO 采用 组内相对奖励 计算优势估计，更符合奖励模型的可比性质

KL 正则化方式不同：

- PPO 在奖励中添加 KL 惩罚项；
- GRPO 直接在损失函数中加入 KL 散度，使其约束更稳定。

目标函数不同：

- PPO 直接优化单个样本的概率比；

- GRPO 计算 组内所有输出的平均值 作为优化目标，使训练更加稳定。

2.2 GRPO的使用方式

Outcome Supervision RL with GRPO

结果监督（Outcome Supervision）仅在输出完成后提供奖励，并不考虑输出过程中的细节。

通过对奖励进行标准化，可以消除奖励的尺度影响，使训练更稳定。

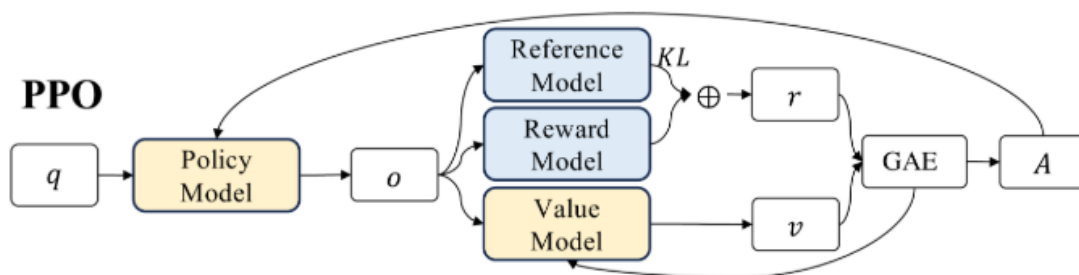
这种方法适用于短文本或简单任务，但对于复杂推理任务，可能监督信号不足。

Process Supervision RL with GRPO

果监督方法仅在输出结束时提供奖励，对于数学推理等复杂任务来说，可能不够充分或高效。因此，我们采用过程监督（Process Supervision），在推理的每个步骤结束时提供奖励。

Iterative RL with GRPO

这里通俗解释一下 Iterative RL



图中Reference Model来自训练前的原模型；Reward Model来自预先训练好的判别模型（奖励模型），这两个都是冻结的。随着训练的进行，Policy Model和Value Model的效果变得越来越好，Policy Model和Value Model与原来冻结的Reference Model和Reward Model的差距变得越来越大，Policy Model输出的结果也与原始冻结的Reference Model的输出差距越来越大，那么此时使用Reference Model和Reward Model来评估效果的效果评估的差距也越来越大。

而我们希望Reward Model能够在Policy Model更新的时候也提升一下能力，即在模型训练的时候也训练一下Reward Model。即可以在训练10%的时间里将Reward Model当作Policy Model，把Policy Model当作Reward Model，两者交换位置交替训练。

随着强化学习训练的进行，旧的奖励模型可能无法有效监督当前的策略模型。因此，我们提出了一种迭代强化学习方法（Iterative RL with GRPO）

流程：

1. **生成新的奖励数据集：**使用策略模型采样数据，构建新的奖励模型训练集。
2. **持续训练奖励模型：**采用重放机制（Replay Mechanism），将 10% 的历史数据与新数据混合训练奖励模型。
3. **更新参考模型：**将当前策略模型设为新的参考模型。
4. **使用新奖励模型继续训练策略模型。**

为什么要迭代训练？

- 训练初期，奖励模型是静态的，但随着策略模型优化，数据分布可能发生变化，导致奖励模型对新策略的指导效果下降。

- 通过迭代训练，奖励模型可以持续适应策略模型的变化，保持监督信号的有效性。

2.3 讨论

Insights of Reinforcement Learning

在本节中，我们提供了一个统一的范式来分析不同的训练方法，例如 SFT（监督微调）、RFT（奖励微调）、DPO（直接偏好优化）、PPO（近端策略优化）、GRPO（梯度正则化策略优化），并进一步进行实验来探索该范式中的关键因素。通常，一个训练方法相对于参数 θ 的梯度可以表示为：

$$\nabla_{\theta} \mathcal{J}_{\mathcal{A}}(\theta) = \underbrace{[E_{(q,o) \sim \mathcal{D}}]}_{\text{Data Source}} \left[\frac{1}{|o|} \sum_{t=1}^{|o|} \underbrace{GC_{\mathcal{A}}(q, o, t, \pi_{rf})}_{\text{Gradient Coefficient}} \nabla_{\theta} \log \pi_{\theta}(o_t | q, o_{<t}) \right].$$

在此公式中，存在三个关键组成部分：

- 数据源 (Data Source) \mathcal{D}** 确定训练数据的来源，即模型的输入数据分布。这可能包括人工标注数据、采样数据或通过不同策略生成的数据。
- 奖励函数 (Reward Function) π_{rf}** 提供训练中的奖励信号，决定模型如何优化策略。这可以是基于人类偏好训练的奖励模型，或者是从外部环境反馈得到的奖励。
- 算法 (Algorithm) \mathcal{A}** 处理训练数据和奖励信号，计算梯度系数 $GC_{\mathcal{A}}$ ，从而确定对数据的惩罚或强化程度。不同的训练方法主要在于如何构造 $GC_{\mathcal{A}}$ ，即如何调整梯度方向和幅度。

| Methods | Data Source | Reward Function | Gradient Coefficient |
|------------|---|-----------------|----------------------|
| SFT | $q, o \sim P_{sft}(Q, O)$ | - | 1 |
| RFT | $q \sim P_{sft}(Q), o \sim \pi_{sft}(O q)$ | Rule | Equation 10 |
| DPO | $q \sim P_{sft}(Q), o^+, o^- \sim \pi_{sft}(O q)$ | Rule | Equation 14 |
| Online RFT | $q \sim P_{sft}(Q), o \sim \pi_{\theta}(O q)$ | Rule | Equation 10 |
| PPO | $q \sim P_{sft}(Q), o \sim \pi_{\theta}(O q)$ | Model | Equation 18 |
| GRPO | $q \sim P_{sft}(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta}(O q)$ | Model | Equation 21 |

Why RL Works?

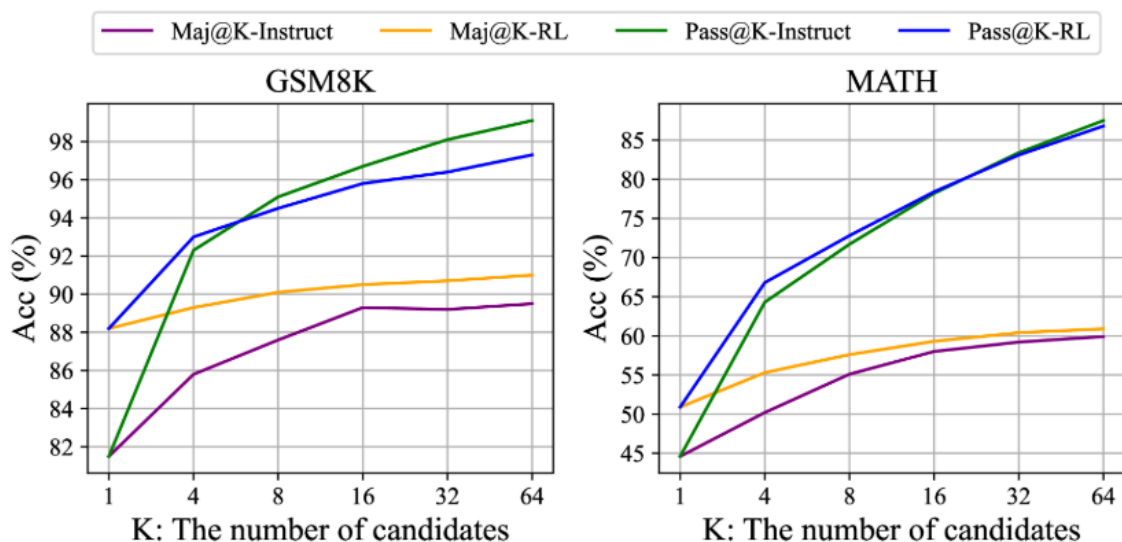


图 7: SFT 和 RL DeepSeekMath 7B 在 GSM8K 和 MATH 上的 Maj@K 和 Pass@K (温度 0.7)。值得注意的是, RL 增强了 Maj@K 但不增强了 Pass@K。

如图 7 所示, RL 增强了 Maj@K 的性能, 但 Pass@K 没有。这些发现表明, RL 通过使输出分布更加稳健来提高模型的整体性能, 换句话说, **这种改进似乎归因于提高 TopK 的正确响应, 而不是基本能力的增强。**

同样地 (Wang 等人, [2023 年一](#)) 发现了 SFT 模型内推理任务中的**错位问题**, 表明可以通过一系列偏好对齐策略来提高 SFT 模型的推理性能

强化学习 (RL) 主要通过**调整输出分布**来提升模型的最终决策稳定性, 而不是让模型学习更多的知识。它让正确答案在 Top-K 选项中更有可能被选中, 而不是提高所有答案的质量。这与之前的研究相符合, 即**偏好对齐 (preference alignment) 可以优化推理任务中的模型表现。**

3. DeepSeek-R1

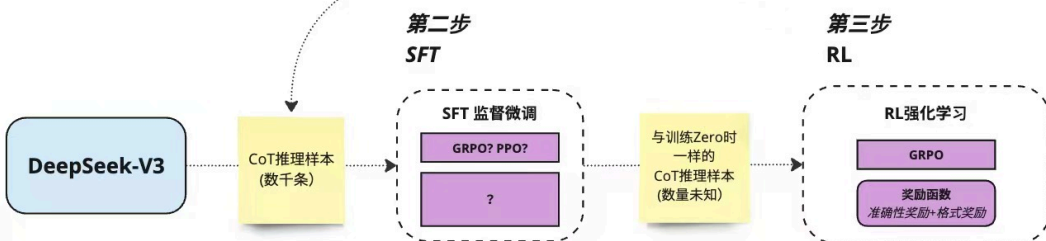
论文地址: [DeepSeek-R1: 通过强化学习激励 LLM 中的推理能力](#)

论文导读: https://www.bilibili.com/video/BV1EmF9e6EdG/?share_source=copy_web

第一步
DeepSeek-R1-Zero



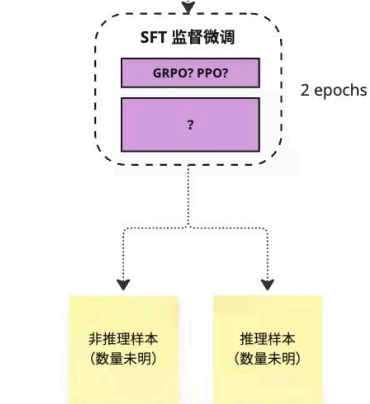
部分样本来自于R1-Zero的生成



从此checkpoint生成数学、代码等逻辑推理样本

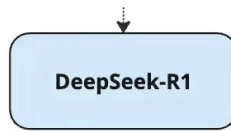


第四步
SFT



第五步
RL





图注：源自导读视频中作者的总结（见评论区置顶）：https://www.bilibili.com/video/BV1EmF9e6EdG/?share_source=copy_web

3.1 贡献 Contributions

后训练：在基础模型上进行大规模强化学习

- 在DeepSeek-v3的基础上直接使用 RL 而没有预先经过 SFT 从而训练出的DeepSeek-R1-Zero。这种直接的方式可以让一个基座模型产生CoT 思维链的能力，从而解决许多复杂问题。DeepSeek是第一个公开大规模实验以验证LLM推理可用纯粹通过RL来提升而无需SFT的，这标志着研究界的一个重要的里程碑。
- 该文介绍了一个用来训练出DeepSeek-R1的pipeline。其包含两个阶段的RL与两个阶段的SFT。

知识蒸馏：小模型也可以很强大

- 我们证明，较大模型所拥有的推理方式（例如CoT等）也可以被蒸馏到小的模型中去。与通过在小模型上使用RL训练得到的推理方式相比，前者效果更好。开源DeepSeek-R1及其API将使研究社区在未来能够蒸馏出更好的小模型。
- 将阿里和meta训练到的模型 Qwen2.5 和 Llama3 通过R1进行蒸馏可以超越，它们同系列中更大的模型的能力。

3.2 方法 Approach

3.2.1 Overview

此前所有工作都使用大量的标注的有监督的数据。在本项工作中证明，即使不使用SFT，也可以通过大量的RL来显著提高推理能力。此外，模型的推理能力还可以被进一步提升，通过少量数据的SFT

3.2.2 DeepSeek-R1-Zero：基础模型上的强化学习

用纯粹的强化学习来优化模型的推理方式。这里DeepSeek-R1-Zero是在以DeepSeek-v3(相当于GPT-4o)为基础模型进行不包含SFT，**没有任何监督数据**的RL。

3.2.2.1强化学习算法 Reinforcement Learning Algorithm

使用DeepSeek与24.2月份贡献的强化学习算法GRPO，GRPO介绍见前文

3.2.2.2 奖励建模 Reward Modeling

奖励是训练信号的来源，它决定了RL的优化方向。在R1-Zero的训练期间，我们采用了rule-based基于规则的奖励系统，包含两种类型的奖励：

- 准确率奖励：准确率奖励模型用于评估响应是否正确。
例如，对于具有确定性结果的数学问题，模型需要以指定格式（例如，在框内）提供最终答案，从而实现可靠的基于规则的正确性验证。同样，对于 LeetCode 问题，可以使用编译器根据预定义的测试用例生成反馈。
- 格式奖励：除了准确性奖励模型外，我们还采用了格式奖励模型，该模型强制模型将其思考过程置于 `<think>` 和 `</think>` 标签之间。

在开发DeepSeek-R1-Zero时，我们使用的奖励系统是基于规则的函数而非神经网络模型。因为我们发现神经网络奖励模型在大规模强化学习中可能会遭遇奖励欺骗（Reward Hacking），重新训练奖励模型需要额外的训练资源，并且使整个训练pipeline复杂化。

“奖励欺骗（Reward Hacking）”指的是，在大规模强化学习过程中，模型可能会找到某种“漏洞”来最大化奖励，而不是学到真正有价值的策略。这意味着：

- 模型可能会生成看似符合奖励函数但实际上不合理或低质量的输出。
- 由于强化学习依赖奖励信号进行优化，如果奖励模型本身存在偏差，训练出来的模型也会继承这些偏差，甚至放大它们。
- 在大规模训练中，这个问题**尤其严重**，因为模型可能会利用训练数据中的模式“投机取巧”，而不是学习通用能力。

Neural Reward Mode 需要不断进行重新训练（retraining）：

- 由于奖励模型本身是一个神经网络，它可能需要定期更新，以适应强化学习过程中策略的变化。

3.2.2.3 训练模板 Training Template

用户和 Google 助理之间的对话。用户提出问题，Google 助理解决该问题。
助手首先在脑海中思考推理过程，然后为用户提供
答案。推理过程和答案包含在 <think> </think> 中，并且
<answer> </answer> 标签，即<think>这里的推理过程 </think>
<answer> 在此处回答 </answer>。 User: 提示符。助理：

表 1：DeepSeek-R1-Zero 的模板。prompt 将替换为 specific reasoning question 在训练期间。

3.2.2.4 DeepSeek-R1-Zero 的性能、自我进化过程 Self-evolution Process 和顿悟时刻 Aha Moment

DeepSeek-R1-Zero 的性能：

| Model | AIME 2024 | | MATH-500 | GPQA Diamond | LiveCode Bench | CodeForces |
|------------------|-----------|---------|----------|--------------|----------------|------------|
| | pass@1 | cons@64 | pass@1 | pass@1 | pass@1 | rating |
| OpenAI-o1-mini | 63.6 | 80.0 | 90.0 | 60.0 | 53.8 | 1820 |
| OpenAI-o1-0912 | 74.4 | 83.3 | 94.8 | 77.3 | 63.4 | 1843 |
| DeepSeek-R1-Zero | 71.0 | 86.7 | 95.9 | 73.3 | 50.0 | 1444 |

表2：DeepSeek-R1-Zero 和 OpenAI o1 模型在推理相关基准测试中的比较（pass@1为一次性输出通过率表现，consistent@64为连续生成64次表现）

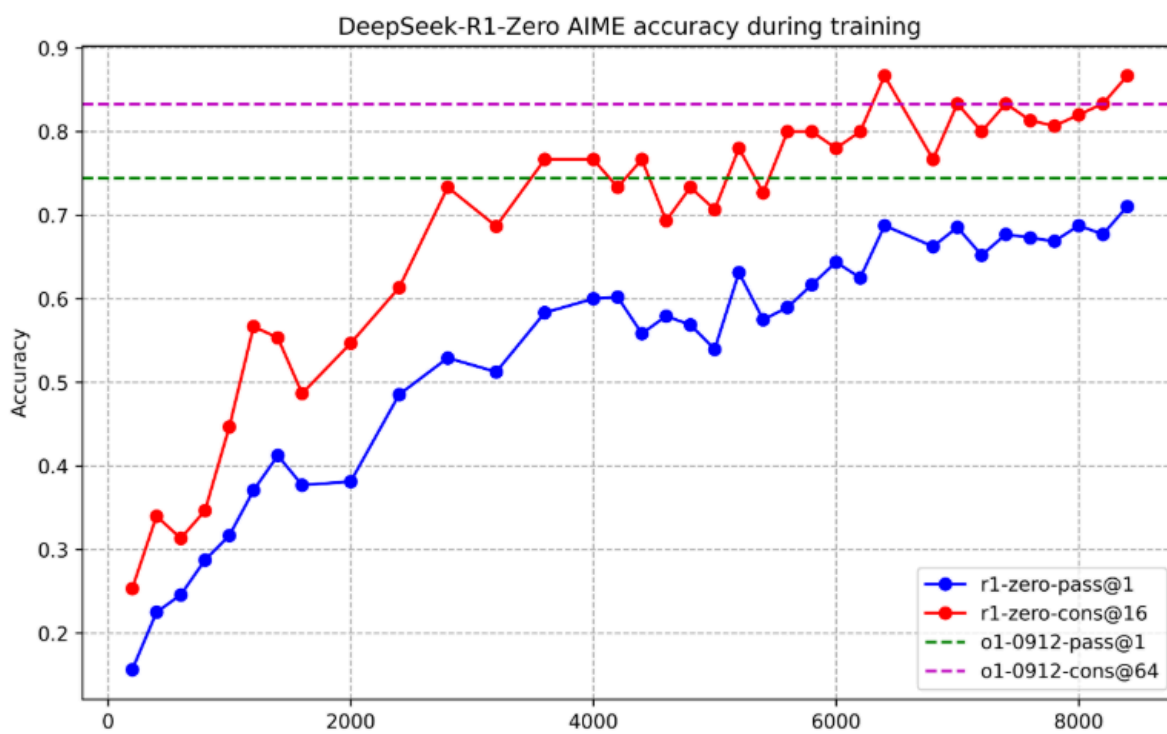


图 2: DeepSeek-R1-Zero 在训练期间的 AIME 准确性。对于每个问题，我们抽样 16 个回答并计算总体平均准确性，以确保评估稳定。

红色是一次性输出准确度，蓝色为输出连续性，可以看到在训练8000步左右R1达到o1的水平。

自我进化过程 Self-evolution Process:

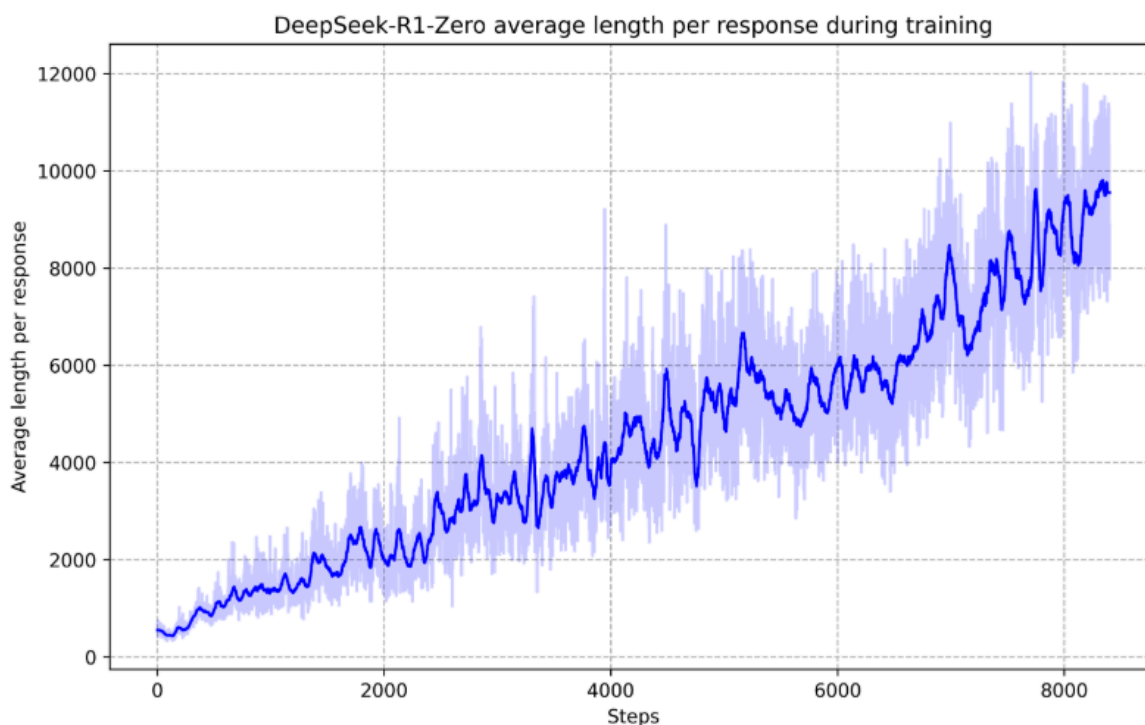


图3: RL 过程中 DeepSeek-R1-Zero 在训练集上的平均响应长度。DeepSeek-R1-Zero 自然而然地学会了用更多的思考时间来解决推理任务。

如图 3 所示，DeepSeek-R1-Zero 的思考时间在整个训练过程中显示出持续的改进。这种改进不是外部调整的结果，而是模型内部的内在发展。DeepSeek-R1-Zero 通过利用增加推理响应时间来进行计算，自然而然地获得了解决日益复杂的推理任务的能力。这种计算范围从生成数百到数千个token，使模型能够更深入地探索和完善其思维过程。

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both ...

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation: ...

...

问题：如果一个 $a > 1$ ，则 的实数解之和 $\sqrt{a - \sqrt{a + x}} = x$ 等于

回应： <think>

求解方程 $\sqrt{a - \sqrt{a + x}} = x$ ，让我们先将两者平方...

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

重新排列以隔离内部平方根项：

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

等等。等。这是我可以在这里标记的顿悟时刻。

让我们逐步重新评估这个，以确定正确的总和是否可以为...

我们从方程式开始：

$$\sqrt{a - \sqrt{a + x}} = x$$

首先，让我们将两边平方：

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

接下来，我可以再次对两侧进行平方，处理方程式： ...

...

表 3：DeepSeek-R1-Zero 中间版本的一个有趣的“顿悟时刻”。该模型学会使用拟人化的语气重新思考。这对我们来说也是一个顿悟的时刻，让我们见证了强化学习的力量和美丽

在 DeepSeek-R1-Zero 的训练过程中观察到的一个特别有趣的现象是“顿悟时刻”的出现。如表 3 所示，这个时刻发生在模型的中间版本中。在这个阶段，DeepSeek-R1-Zero 通过重新评估其初始方法，学会为问题分配更多的思考时间。这种行为不仅证明了模型不断增长的推理能力，也是强化学习如何导致意想不到的复杂结果的迷人例子。

这一刻不仅是模型的“顿悟时刻”，也是观察其行为的研究人员的“顿悟时刻”。它强调了强化学习的力量和美感：我们不是明确地教模型如何解决问题，而是简单地为其提供正确的激励措施，然后它就会自主开发高级问题解决策略。“顿悟时刻”有力地提醒我们，RL 有可能在人工系统中解锁新的智能水平，为未来更加自主和自适应的模型铺平道路。

尽管 DeepSeek-R1-Zero 表现出强大的推理能力，但它面临几个问题。例如，DeepSeek-R1-Zero 正在努力应对可读性差和语言混合等挑战。为了使推理过程更具可读性，我们探索了 DeepSeek-R1，这是一种利用 RL 和人类友好型冷启动数据的方法。

3.2.3 DeepSeek-R1：使用冷启动进行强化学习

受到 DeepSeek-R1-Zero 的可喜结果的启发，自然而然地出现了两个问题：

- 1) 通过将少量高质量数据作为冷启动，是否可以进一步提高推理性能或加速收敛？
- 2) 我们如何训练一个用户友好的模型，该模型不仅产生清晰连贯的思维链（CoT），而且还表现出强大的通用能力？

为了解决这些问题，我们设计了一个pipeline来训练 DeepSeek-R1。该pipeline由四个阶段组成，概述如下。

3.2.3.1 冷启动 Cold Start

与 DeepSeek-R1-Zero 不同，为了防止基础模型出现 RL 训练的早期不稳定冷启动阶段，对于 DeepSeek-R1，我们构建并收集少量长 CoT 数据，以将模型作为初始 RL 参与者进行微调。为了收集这些数据，我们探索了几种方法：以长 CoT 的 few-shot 提示为例，直接提示模型通过反思和验证生成详细的答案，以可读格式收集 DeepSeek-R1-Zero 输出，并通过人工注释者进行后处理来提炼结果。

我们收集了数千个冷启动数据，以微调 DeepSeek-V3-Base 作为 RL 的起点。与 DeepSeek-R1-Zero 相比，冷启动数据的优势包括对人类的可读性，和先前实验得到的性能更好的结论

3.2.3.2 面向推理的强化学习 Reasoning-oriented Reinforcement Learning

在根据冷启动数据微调 DeepSeek-V3-Base 后，我们应用了与 DeepSeek-R1-Zero 中相同的大规模强化学习训练过程。此阶段侧重于增强模型的推理能力，尤其是在推理密集型任务中。

在训练过程中，我们观察到 CoT 经常表现出语言混合，尤其是当 RL 提示涉及多种语言时。为了缓解语言混合问题，我们在 RL 训练期间引入了语言一致性奖励，其计算方式是目标语言单词在 CoT 中的比例。

尽管消融实验表明，这种对齐会导致模型的性能略有下降，但这种奖励与人类的偏好一致，使其更具可读性。最后，我们将推理任务的准确性和语言一致性的奖励结合起来，直接相加形成最终的奖励。然后，我们在微调模型上应用 RL 训练，直到它在推理任务上实现收敛。

3.2.3.3 抑制采样和监督微调 Rejection Sampling and Supervised Fine-Tuning

我们将上一轮面向推理的RL训练收敛时的checkpoint来收集SFT数据，用于下一轮的巡的训练。与主要关注推理的初始冷启动数据不同，此阶段整合了来自其他领域的的数据，以增强模型在写作、角色扮演和其他通用任务方面的能力。

推理数据 Reasoning data

我们通过从上述 RL 训练的权重执行抑制采样来策划推理提示并生成推理轨迹。在上一阶段，我们只包含可以使用基于规则的奖励进行评估的数据。然而，在这个阶段，我们通过整合额外的数据来扩展数据集，其中一些数据通过使用生成奖励模型，将真实和模型预测输入到 DeepSeek-V3 中进行判断。此外，由于模型输出有时混乱且难以阅读，因此我们过滤掉了混合语言、长释义和代码块的思维链。对于每个提示，我们会对多个响应进行采样，并仅保留正确的响应。我们总共收集了大约 600k 个推理相关的训练样本。

非推理数据 Non-Reasoning data

对于非推理数据，如写作、事实 QA、自我认知和翻译，我们采用DeepSeek-V3 pipeline，并复用DeepSeek-V3的SFT数据集的一部分。对于某些非推理任务，我们调用DeepSeek-V3来生成一个潜在的思维链，然后通过提示来回答问题。但是，对于更简单的查询，例如“hello”，我们不提供 CoT 作为响应。最后，我们总共收集了大约 200k 个与推理无关的训练样本。

我们使用上述约 800k 样本的精选数据集对 DeepSeek-V3-Base 进行了两个 epoch 的微调。

3.2.3.4 适用于所有场景的强化学习 Reinforcement Learning for all Scenarios

使用一种两个阶段的强化学习训练：

- 对于推理数据，我们遵循 DeepSeek-R1-Zero 中的RL方法，利用基于规则的奖励函数来指导数学、代码和逻辑推理领域的学习过程。
- 对于一般数据，我们采用奖励模型来捕捉复杂和细微场景中的人类偏好。

我们以 DeepSeek-V3 pipeline为基础，采用类似的偏好对和训练提示分布。

3.2.4 Distillation：为小模型赋能推理能力

我们直接对 Qwen 等开源模型进行了蒸馏微调，使用通过 DeepSeek-R1 精选的 800k 样本，这种简单的蒸馏方法显着增强了较小模型的推理能力。

对于蒸馏模型，我们只应用 SFT，不包括 RL 阶段，即使合并 RL 可以大大提高模型性能。我们在这里的主要目标是证明蒸馏技术的有效性。

3.4 讨论

3.4.1 知识蒸馏与强化学习

通过蒸馏 DeepSeek-R1，小模型可以取得令人印象深刻的结果。然而，仍然剩下一个问题：该模型能否通过论文中讨论的大规模 RL 训练实现相当的性能，而无需蒸馏？

| Model | AIME 2024 | | MATH-500 | GPQA Diamond | LiveCodeBench |
|------------------------------|-----------|---------|----------|--------------|---------------|
| | pass@1 | cons@64 | pass@1 | pass@1 | pass@1 |
| QwQ-32B-Preview | 50.0 | 60.0 | 90.6 | 54.5 | 41.9 |
| DeepSeek-R1-Zero-Qwen-32B | 47.0 | 60.0 | 91.6 | 55.0 | 40.2 |
| DeepSeek-R1-Distill-Qwen-32B | 72.6 | 83.3 | 94.3 | 62.1 | 57.2 |

表6：蒸馏模型和 RL 模型在推理相关基准上的比较。

为了回答这个问题，我们使用数学、代码和 STEM 数据在 Qwen-32B-Base 上进行了大规模 RL 训练，训练了超过 10K 个步骤，从而产生了 DeepSeek-R1-Zero-Qwen-32B。表 6 所示的实验结果表明，经过大规模 RL 训练后，32B 基础模型的性能与 QwQ-32B-Preview 相当。但是，从 DeepSeek-R1 中提炼出来的 DeepSeek-R1-Distill-Qwen-32B 在所有基准测试中的表现明显优于 DeepSeek-R1-Zero-Qwen-32B。

因此，我们可以得出两个结论：首先，将更强大的模型蒸馏成更小的模型会产生极好的结果，而依赖于本文提到的大规模 RL 的较小模型需要巨大的计算能力，甚至可能无法达到蒸馏的性能。其次，虽然蒸馏策略既经济又有效，但超越智能界限可能仍然需要更强大的基础模型和更大规模的强化学习。

3.4.2 未成功的尝试

流程奖励模型（PRM）

PRM 的核心思想

- PRM（Process Reward Model）是一种用于推理任务的强化学习方法，**通过奖励细粒度的推理步骤来引导模型学习更好的推理路径。**
- 目标是鼓励模型采取更合理的解题路径，而不仅仅是最终答案正确。

PRM 的三个主要局限

1. 难以定义细粒度推理步骤

- 在一般推理任务中，拆解出清晰的、可学习的中间步骤并不容易。
- 例如，数学推理可能有多种不同的解题路径，难以确定哪种路径才是“最优的”。

2. 难以准确评估中间步骤的正确性

- 需要判断模型当前的中间推理步骤是否正确，这在实践中是一个很大的挑战：
 - **自动标注 (auto-annotation)**：使用模型来自动生成推理步骤的正确性标签，可能会产生错误或偏差。
 - **人工标注 (manual annotation)**：需要大量人工干预，难以扩展到大规模数据集。

3. Reward hacking 问题 + 训练成本

- PRM 可能会导致奖励欺骗 (Reward Hacking)
 - 例如，模型可能学会“优化奖励”而不是“真正改进推理”。
- 需要不断重新训练奖励模型
 - 训练和维护奖励模型需要额外的计算资源，增加了训练的复杂性。

蒙特卡洛树搜索 (MCTS)

MCTS 的核心思想

- 受 AlphaGo (Silver et al., 2017b) 和 AlphaZero (Silver et al., 2017a) 启发，MCTS 被用于在推理任务中增强计算的可扩展性。
- 核心思路：
 - **将复杂问题拆解为较小的部分**，然后逐步探索解空间。
 - **引导模型进行搜索**，让它生成多个标签，对应于搜索过程中需要的特定推理步骤。
 - **使用 MCTS 搜索答案**，再利用搜索得到的答案训练一个**价值模型 (value model)**，并不断优化整个过程。

MCTS 在推理任务上的挑战

1. 搜索空间远大于棋类游戏

- 在国际象棋或围棋中，搜索空间有限，状态转换明确。
- 但在语言任务中，**token 的组合方式呈指数级增长**，导致搜索空间**比棋类游戏大得多**，难以进行高效搜索。
- 解决方案：限制每个搜索节点的扩展数量（但可能会导致局部最优问题）。

2. 价值模型的训练困难

- 价值模型 (Value Model) 决定了搜索过程中哪些路径应该被优先考虑。
- 但训练一个细粒度的价值模型本身就非常困难，特别是在语言任务中，价值模型可能会**误导搜索过程**，从而影响最终生成质量。

3. 难以复制 AlphaGo 的成功经验

- AlphaGo 之所以成功，是因为**价值模型的逐步优化**，让搜索变得越来越精准。
- 但在**自然语言任务**中，由于搜索空间过大、奖励信号复杂，**这一策略难以复现**，导致 MCTS 在增强推理能力方面的效果有限。